DataGeneral

INTRODUCTION TO THE ECLIPSE™ REAL TIME DISK OPERATING SYSTEM

093-000132-00

INTRODUCTION TO THE ECLIPSE™ REAL TIME DISK OPERATING SYSTEM

093–000132–00

Ordering No. 093-000132

Printed in the United States of America
Rev. 00, May 1975

This document is intended for ECLIPSE™* system users. It was derived from 093-000083-04; which is primarily for NOVA® ** system users. There may still be references to NOVA equipment documents, and processes within this manual; if so, be assured that the references also apply to ECLIPSE systems.

Revision History:

093-000083

Original Release - September 1972
First Revision - October 1972
Second Revision - November 1972
Third Revision - October 1973
Fourth Revision - January 1975

093-000132

Original Release - May 1975

*ECLIPSE is a trademark of Data General Corporation, Southboro, Massachusetts.
**NOVA is a registered trademark of Data General Corporation, Southboro, Massachusetts.

TABLE OF CONTENTS

*RDOS overview topics

LIST OF ILLUSTRATIONS

HOW TO USE THIS MANUAL

This manual uses modular instruction techniques to present a complex topic, the Real Time Disk Operating System, in a simple, step-by-step fashion. Each topic is presented as a module comprised of a single page of text accompanied by a single-page illustration. There is a loose logical progression in the presentation of topics, yet each topic is truly self-contained. Thus there is no need to digest all the material in a single sitting, and you may indeed read the modules selectively according to your particular interests. If you would like a broad overview of the operating system, note that you may read only those modules whose titles are starred in the table of contents.

No manual of this kind can present any topic in depth, and so after you have finished this primer you should consult one or more of the publications listed in the back of this manual following the glossary of terms. This list of publications gives you the titles and numbers of all DGC manuals describing RDOS and major utilities available under RDOS. Additionally, this list gives an abstract describing what you may expect to find in each of these publications.

RDOS Features:

- dual programming systems
- dual-processor/shared-disk support
- modular multi-task monitor
- partially core-resident, partially disk resident operating system
- user program swaps, chains and overlays
- spooling of output to peripherals
- powerful disk file structures, disk partitions and subdirectories
- buffered and non-buffered device or file I/O
- support for high level languages with real time extensions
- interactive and batch job processing

AN INTRODUCTION TO RDOS

Data General's RDOS is a Real Time Disk Operating System. RDOS is real time oriented since it can schedule and allocate program control to many different subprogram tasks to provide simultaneous use of system resources, maximizing throughput and insuring efficiency and economy of total program operation.

A modern real time operating system must be geared to change and diversity. The RDOS system itself can exist in an almost unlimited variety of machine configurations; different installations will typically have different configurations as well as different applications. Moreover, the configuration and scope of the project at a given installation may frequently change. Thus the operating system must be adaptable to a number of different environments. All of this puts a premium on the system modularity and flexibility which has been designed into RDOS.

To obtain the full capabilities of the Real Time Disk Operating System, the user requires only a NOVA® ** or ECLIPSE™ *** computer with 16K words of core memory, a console Teletype® * or Data General Video Display, and a disk. In addition to this minimum machine configuration, RDOS supports additional core storage (in excess of 65K words), four million words of fixed head disk, eight disk cartridges or pack drives, sixteen magnetic tape transports (7 and/or 9 track) and/or cassette units, card readers, line printers, communication equipment, inter-processor links, digital plotters, and multiprocessor communication adapters.

RDOS can be used both interactively from a console keyboard and in BATCH mode. BATCH job streams may be obtained from card readers, disk files, cassette files, or magnetic tape files.

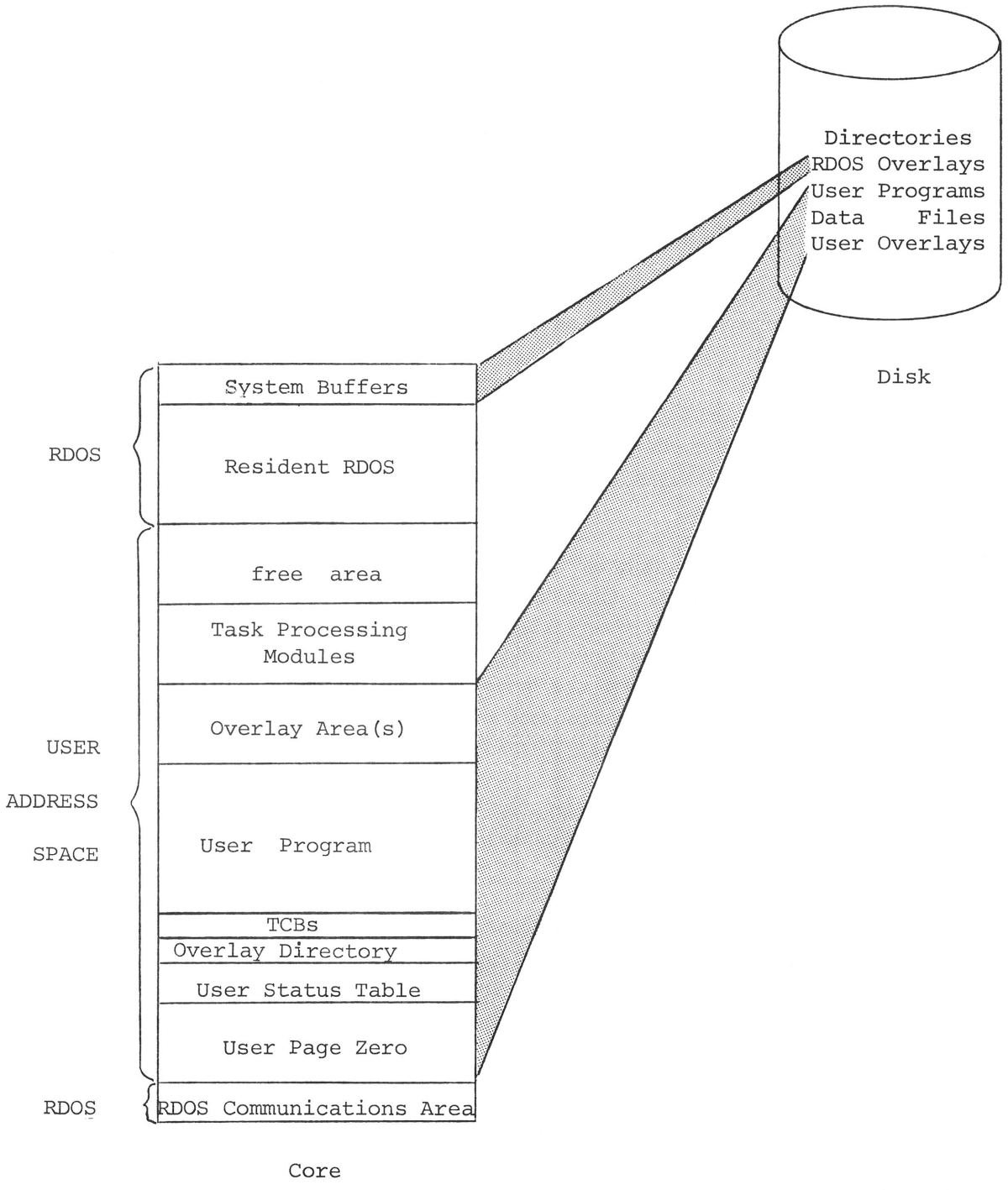
In addition to running multitask user programs, the NOVA 840 and ECLIPSE computers under mapped RDOS can support dual programming environments in configurations with greater than 65K words of main memory. Dual processors sharing disk file space can be used for all demanding real time applications where redundancy is important.

---

*Teletype is a registered trademark of Teletype Corporation, Skokie, Illinois.

**NOVA is a registered trademark of Data General Corporation, Southboro, Massachusetts.

***ECLIPSE is a trademark of Data General Corporation, Southboro, Massachusetts.



RDOS Organization

RDOS ORGANIZATION

The RDOS executive constitutes the main framework of the operating system, and it must be resident in main memory before any continuous and coordinated processing can occur. Functions performed by this resident portion of RDOS include interrupt processing, overlay and buffer management, system call processing, and device interrupt servicing. Other modules of the system are brought into main memory from disk storage as they are required to perform specific functions such as full or partial system and device initializations, file maintenance operations such as opening, closing, renaming, or deleting files, and spooling control.

Mapped RDOS (MRDOS) supports dual programming and mapped memory addressing. However, the single program environment with unmapped addressing provides the simplest study of RDOS organization. In such a basic system, user page zero includes address 16 (USP) and then extends from location 20 through 377. USP is a special location preserved by RDOS whenever program control is transferred from one task to another. Location 17 is reserved by the operating system for use in system call processing (all system calls cause an indirect call to the call processor via location 17).

The RDOS User Status Table begins at the first address of normally relocatable memory, 400_8. This table extends from 400 through 423 octal, and it contains information describing the user program. This information includes the user program's length, the number of tasks and active I/O channels specified by the program, and other permanent and variable information.

Following the User Status Table are the Overlay Directory (if user overlay areas are defined) and the Task Control Block (TCB) pool. TCBs are required by the system to record current state variable information pertaining to each active task within the program. Following this pool is the user program proper. One or more user overlay areas may be interspersed throughout the user program to receive disk-resident portions of the user program when they are requested by the program. Finally, following the user program are the system library modules required to support multitask calls issued by the program. Also loaded from SYS.LB is the multitask scheduler which switches control from task to task during the operation of the program.

```
SYSGEN
SYSGEN REV 4.00
VALID ANSWERS ARE IN PARENTHESIS RESPOND ACCORDINGLY
NOVA ("0") OR ECLIPSE ("1")?  1
MAPPED SYSTEM?  ("0" = NO  "1" = YES)  1
MAXIMUM NUMBER OF CHANNELS BACKGROUND WILL USE (1-N)  10
MAXIMUM NUMBER OF CHANNELS FOREGROUND WILL USE (0-N)  10
NUMBER OF FIXED HEAD DISK CONTROLLERS (0-2)  0
NUMBER OF DISK PACK CONTROLLERS (0-2)  1
DEVICE PRIMARY ("0") OR SECONDARY ("1")?  0
ENTER NUMBER OF DEVICES FOR CONTROLLER #1 (1-4)  1
DUAL PROCESSORS (IPB)?  ("0" = NO  "1" = YES)  0
ENTER NUMBER OF STACKS (1-10)  5
ENTER NUMBER OF EXTRA CELLS (0-N)  8
TUNING?  ("0" = NO  "1" = YES)  1
SHALL TUNING BE WITH ("1") OR WITHOUT ("0") OVERLAY REPORT?  1
ENTER NUMBER OF EXTRA BUFFERS REQUIRED (0-N)  6
MAXIMUM NUMBER OF SUB-DIRECTORIES/SUB-PARTITIONS
ACCESSIBLE AT ONE TIME (0-32)  4
ENTER NUMBER OF CONTROLLERS FOR MTA (0-2)  1
DEVICE PRIMARY ("0") OR SECONDARY ("1")  0
ENTER NUMBER OF DEVICES FOR CONTROLLER #1 (1-8)  2
ENTER NUMBER OF CONTROLLERS FOR CTA (0-2)  0
AUTO RESTART ON POWER FAIL?  ("0" = NO  "1" = YES)  1
OPERATER MESSAGES ? ("0" = NO  "1" = YES)  0
RTC?  ("0" = NO  "1" = YES)  1
ENTER RTC FREQ (1 = 10HZ, 2 = 50HZ, 3 = 60HZ, 4 = 100HZ, 5 = 1000HZ)  1
ENTER NUMBER OF PTR (0-2)  1
ENTER NUMBER OF PTP (0-2)  0
ENTER NUMBER OF LPT (0-2)  1
ENTER COLUMN SIZE FOR DEVICE #1 ( 80 or 132)  80
ENTER NUMBER OF CDR (0-2)  0
ENTER NUMBER OF PLT (0-2)  0
ENTER NUMBER OF MCA (0-2)  0
QTY?  ("0" = NO  "1" = YES) 0
SECOND TTY?  ("0" = NO  "1" = YES)  1
CORE DUMP FACILITY? ("0" = NO "1" = YES) 1
R
```

Sample System Generation Dialogue

SYSTEM GENERATION

In real time operating systems, individual user requirements may vary within an installation either because of differing hardware needs or because of dissimilarities in system features. These differences may take the form of different applications, different configurations of standard hardware, special process input/output hardware, the size of main memory, system throughput and priority considerations, and the availability of hardware memory management.

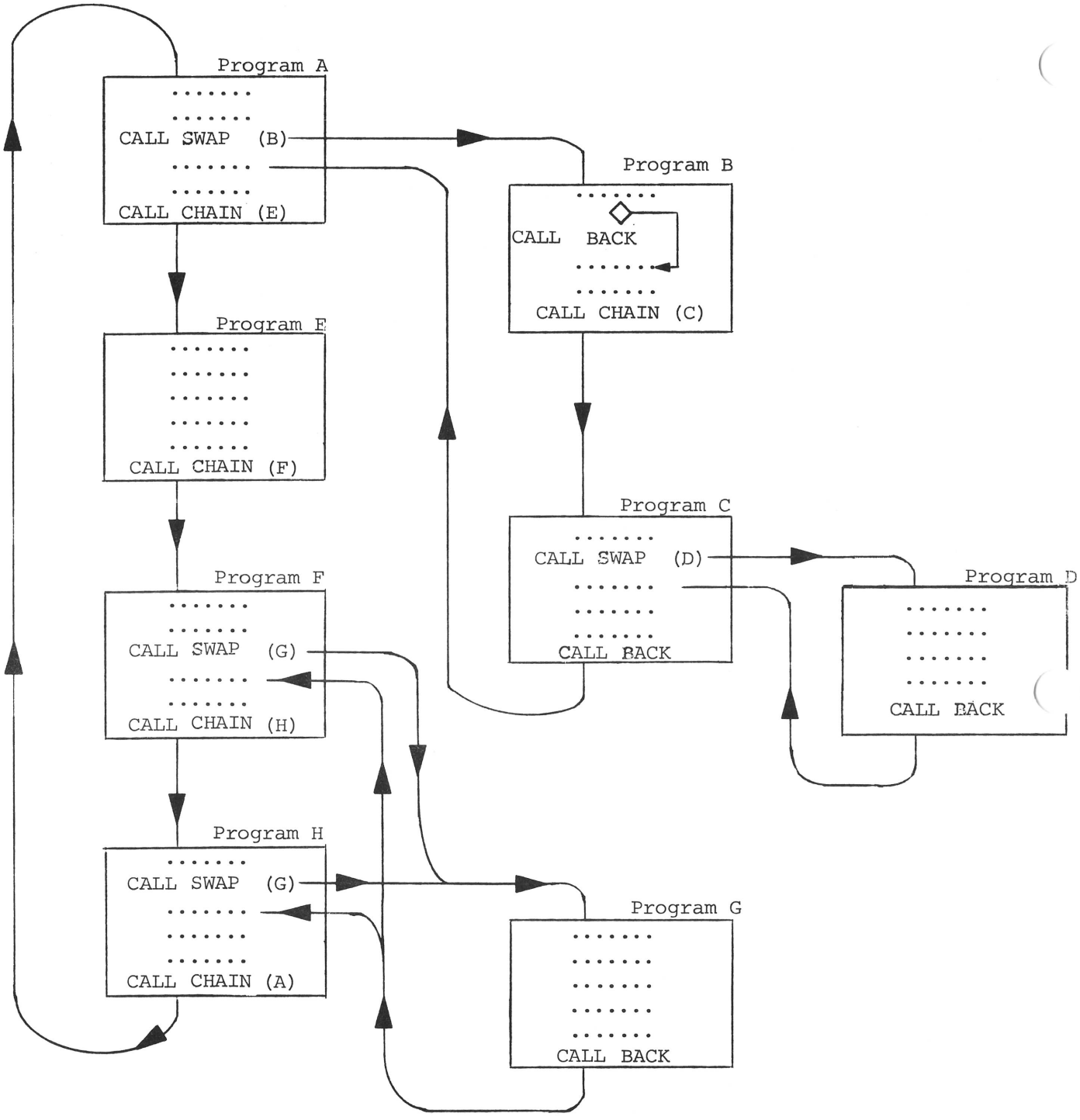
In essence, each system must be tailored to the specific requirements and input/output hardware configuration of that installation. The tailoring function is defined as system generation, and is performed initially with the support of a generalized starter operating system provided by Data General. The system generation process provides the means to create a monitor composed of programs and subroutines written by DGC and optionally by the user. The end product of system generation is a disk resident operating system which is custom built to provide an efficient executive for a specific machine configuration and system operational requirements.

The efficiency of any operating system is enhanced by the selection of those system components which most closely match the requirements of the program environment. To aid this enhancement process, RDOS provides a facility called tuning report generation which enables the efficiency of the system to be monitored so that suitable adjustments to the system can be made. In fact, RDOS is capable of self-tuning, generating tuning reports and selecting system components which will produce a more efficient system. Optimum efficiency, however, can be obtained only by the user's personal participation in the system generation process.

The modular design of RDOS and the availability of numerous system features make it possible to configure multiple RDOS systems. Each system can be tailored to the individual applications requirements. The resulting customized system can support the following user environments:

- single program with single or multitasking
- dual programs, each with single or multitasking
- dual programs operating in a hardware protected system
- dual processors sharing disk storage
- multiple processors with or without shared disks

Each disk may contain one or more of these tailored systems, and the RDOS bootstrap program, HIPBOOT, permits each of these systems to be placed into execution at will.



FORTRAN Swaps and Chains

PROGRAM SWAPS AND CHAINS

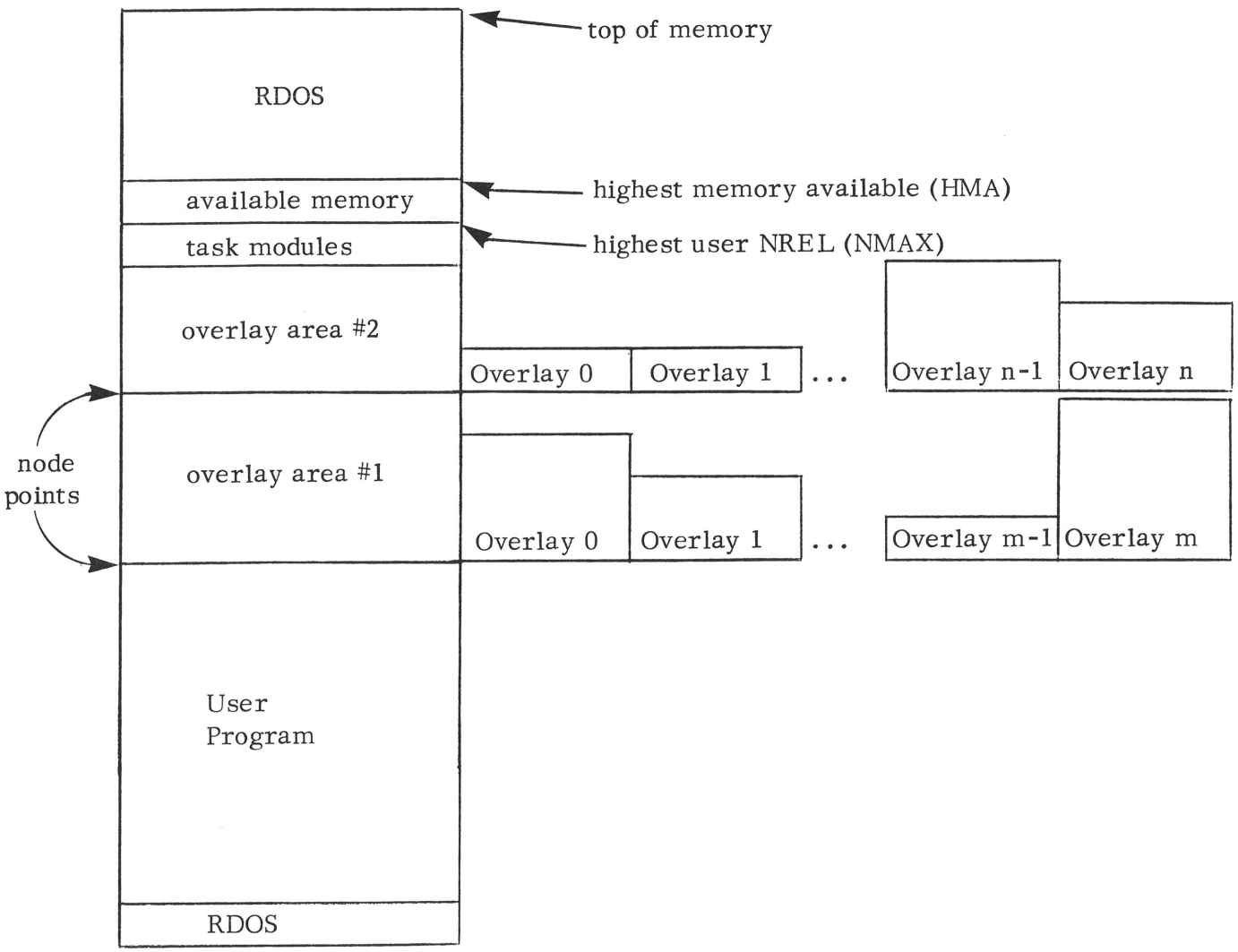
Any program running under RDOS can suspend its own execution and either invoke another distinct program or call for a new section of itself. The current program is overwritten in both cases.

Every new program (or section of the same program) which is invoked must exist on the disk in core image format. Both program swaps and chains contain core images extending from address 16 (USP) through the highest address in the user program (NMAX). Not included in this area is blank or unlabeled common in the case of FORTRAN programs.

The program chain facility permits programs to be run which require more core storage than is ever available at one time. To use chaining, a program must be written in serially executable segments, each of which calls the next segment.

The program swap facility, by contrast, permits distinct programs to call one another in much the same manner as subroutines are called. The primary difference is the size of the routine and the manner of argument passing. Whereas subroutines are always smaller than the total amount of resident core, program swaps are as large as total user program space (or as large as multiples of user address space if chaining is performed). The program whose core images are overwritten during a program swap operation is stored temporarily on disk. Information saved enables the program to be restarted upon return from the swap.

There are two ways that parameters can be passed to program swaps and that results can be returned to the callers. These means are: (1) via disk files with agreed upon names, and (2) via unlabeled common. Communication via both methods is possible with both program swaps and program chains.



Organization of Program with Overlays

USER PROGRAM SEGMENTATION

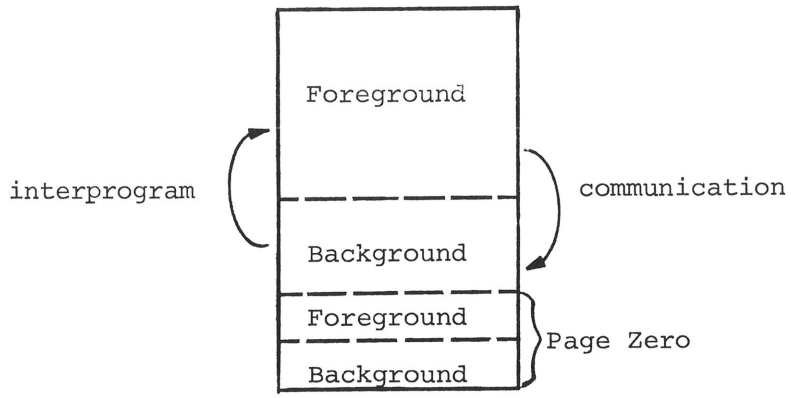
When user address space is not sufficient to contain all the programs which are necessary for a system's operation, some form of program read-in scheme must be employed whereby programs, upon demand, are brought in from disk storage. The RDOS system can reserve portions of user address space for this function, and divides it into fixed-length partitioned core storage areas which form a repository for programs of a limited size. This allows the RDOS user to segment his program into one or more parts which fit into the fixed-size core areas at execution time. These program segments are called user overlays, and are stored on disk in core image format to facilitate rapid loading when their execution is desired.

User overlays differ from program swaps in that user overlays overwrite only a fixed area within a root program which remains active and core resident during the load time. Moreover, user overlays are more flexible than program swaps, since the user is permitted to be executing program segments in one or more currently resident overlays while a non-resident overlay is being loaded by the system. Moreover, if more than one user requests a specific overlay, that overlay will be locked into core memory until all requests for its use have been satisfied. RDOS itself uses overlays to enhance its own operations; these overlays are called system overlays to distinguish them from user overlays.

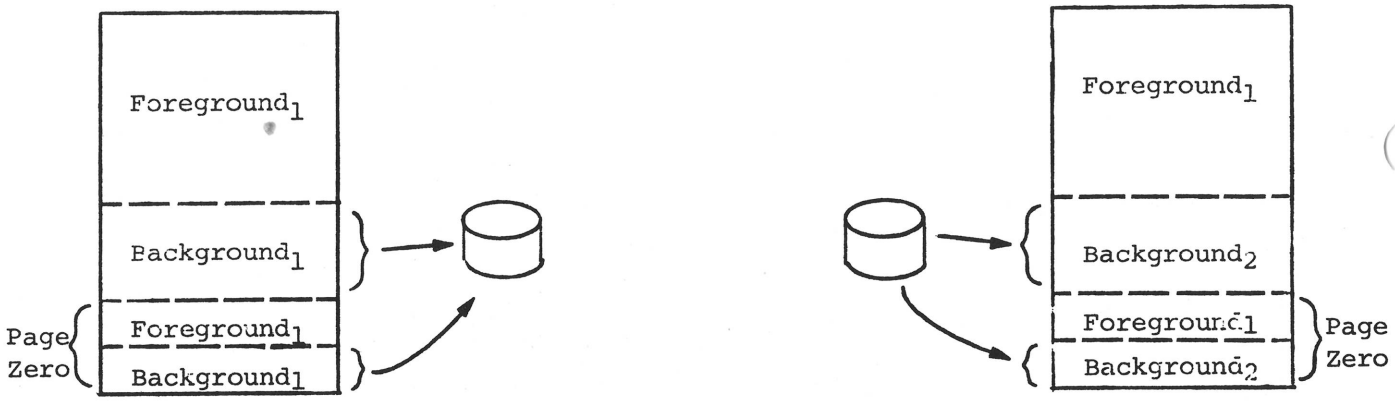
The complete set of user overlays used by a root program is called a user overlay file. Overlay files are organized contiguously. Contiguous file organization and core image format combine to make overlay loading into a root program quick and efficient.

The beginning of each overlay area is called its node point. Associated with each area is a collection of one or more user overlays. The relocatable loader ensures that each overlay area within a root program is made large enough to accommodate the largest overlay which will be loaded into that area. There may be up to 128 user overlay areas within any root program. Each area may receive 256 separate user overlays.

RDOS also provides a utility called the overlay replacement loader, OVLDR, which permits overlays within an overlay file to be replaced by different overlays which have been developed in another program. This replacement can be performed without having to halt the currently executing program which loads overlays from the file, since the overlay replacement is performed only when the file is not being accessed.



The Foreground and Background can intercommunicate,



and the Foreground can checkpoint the Background, if desired.

Foreground-Background Program Communication

FOREGROUND-BACKGROUND PROGRAMMING

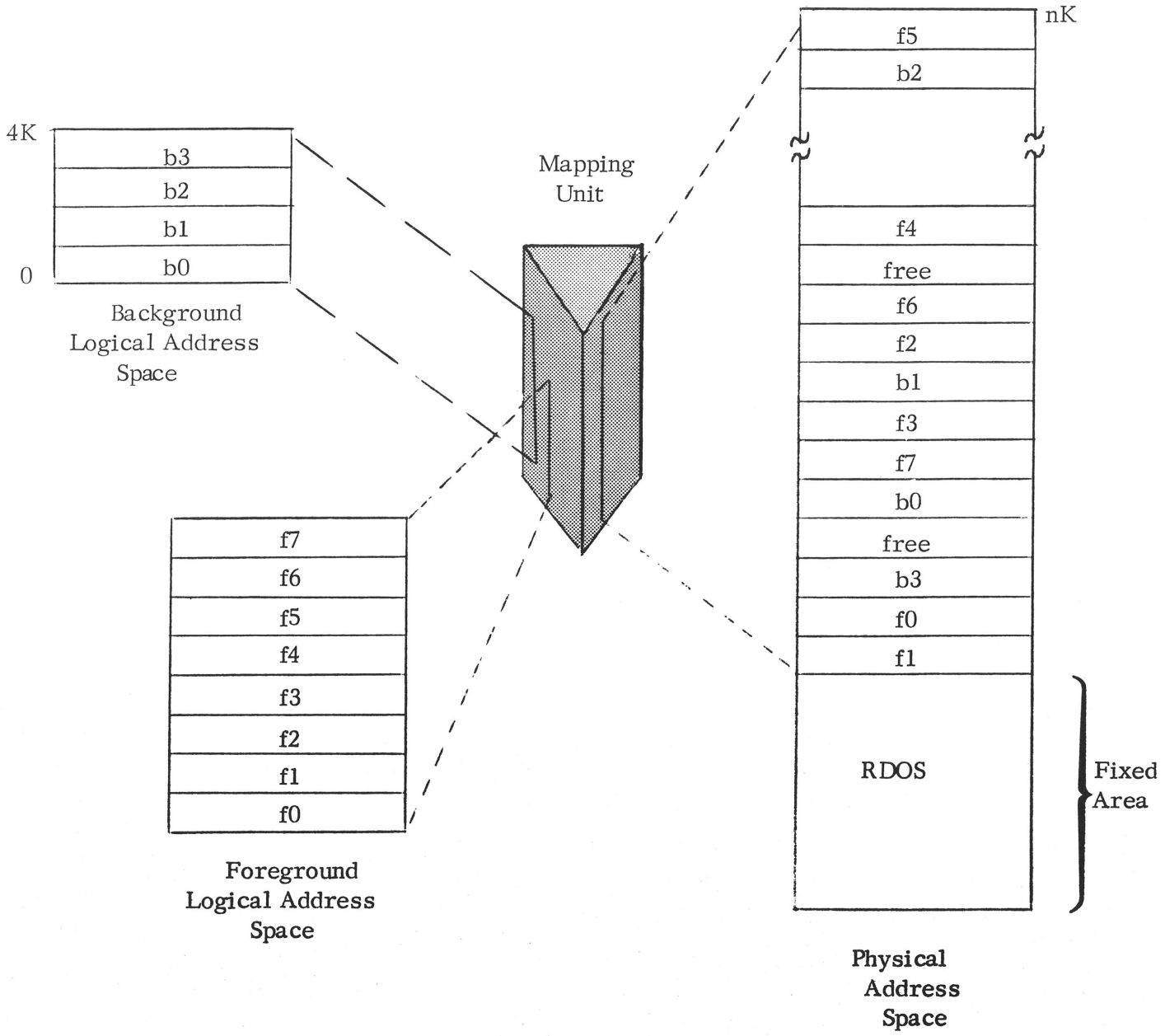
In most current computer installations, real time control and program development functions are performed sequentially. That is, program development must be completed before any real time program may be run. Moreover, once a real time program is in execution, no other program can be run concurrently even though the real time program itself requires only a fraction of available system resources at any given moment. This inefficient use of computer system resources is eliminated if some way is found to permit more than one program to be run concurrently.

To increase system utilization, RDOS permits two programs to be core-resident and run with apparent simultaneity. That is, the operating system switches control between one of two task schedulers according to a pre-determined priority assigned to each by the user. Foreground-background programming permits unrelated collections of tasks to be performed, sharing basic system resources. A typical foreground-background system under RDOS might contain a real time process control program in the foreground area and an assembly, compilation, and payroll program at different times in the background area.

To protect the integrity of both programs, software partitions are created during the relocatable loading of foreground and background programs. There is both a page zero (ZREL) and a non-page zero (NREL) partition. Each indicates the starting address of the foreground program area and is indicated by dashed lines in the illustration on the preceding page.

Although the foreground and background programs are truly independent, the operating system does permit them to act in concert if this is desired. The foreground and background programs can issue messages to one another via certain system calls, and these programs can also intercommunicate via the multiprocessor communications adapter (MCA).

Additionally, mapped systems permit the foreground to send the current background to disk temporarily and to read in a different background program for execution. One example of such an application consists of a data collection program in the foreground and one of several program development utilities in the background. In such an application, the foreground might require occasional use of a data reduction program to be executed in the background. Checkpointing, the practice of interrupting a lower priority background program temporarily with a special, higher priority program, would fulfill this need.



Logical-to-Physical Address Mapping

DUAL PROGRAMMING

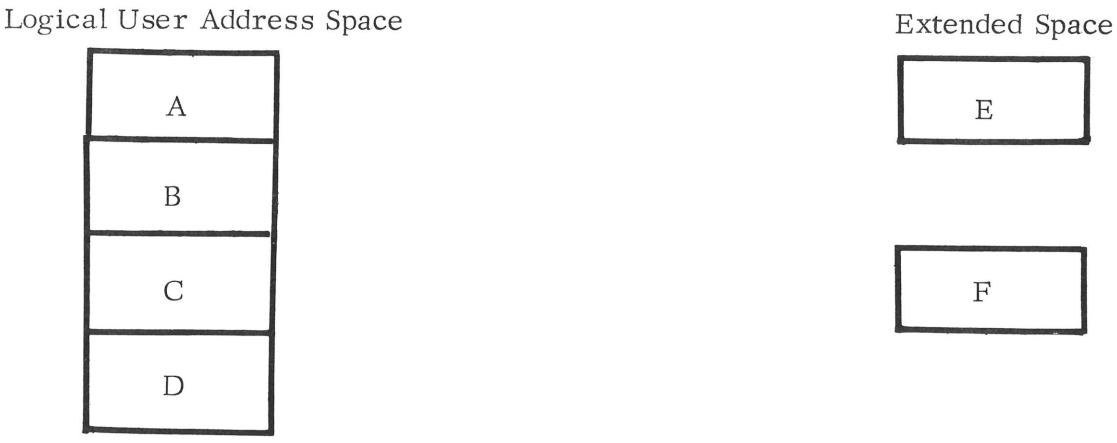
Foreground-background programming with software memory partitions is unquestionably a more efficient use of computer resources than is single program execution. Nonetheless, even this improved scheme has drawbacks which limit its usefulness. The first drawback is that there is no guarantee that if one program experiences failure (e.g., by accessing cells outside its allotted address space) it will not cause the other program to fail also. Secondly, foreground-background programming requires that great care be exercised to ensure that each program be tailored to fit into available memory. Also, page zero must be shared by both the foreground and the background, and this restricts the amount of this resource available to each. Data General Corporation has overcome all of these limitations by developing an option called the Memory Management and Protection Unit, available on NOVA 840 and ECLIPSE computers.

These computers provide a hardware separation of the background, foreground and operating system areas. Moreover, they extend the maximum core configuration for a single CPU from 32K to up to 32K for the resident operating system and up to 32K directly addressable by each background and foreground program. (Maximum address areas are 31K on NOVA MRDOS systems.) In a mapped system, two addressing modes exist. In the first mode, absolute mode, only the lower 32K is directly addressable and the mapping device is not used. RDOS resides in these low physical memory locations and executes in absolute mode.

The second mode is called mapped or user mode. In user mode, up to thirty-two 1024_{10} word blocks of memory are mapped by the management unit so as to produce an apparent (logical) 32K continuous address space. Both the foreground and the background programs execute in mapped mode and are essentially unaware of their actual memory locations.

Any program operating in user mode uses a complete logical address space including its private page zero and extending through its upper memory bound (NMAX). NMAX is determined by the requirements of the individual program and it may extend as high as 32K. The operating system is responsible for assigning free memory from its available pool to each user program prior to its execution. The technique used to manage the mapping unit and the construction of the user program in logical address space is also the responsibility of MRDOS.

While a user program is running, it may communicate with the operating system via all the standard RDOS . SYSTM calls and by a few additional calls designed for mapped systems. Moreover, both system and user devices may use the data channel, and consequently its utilization must be allocated by the operating system.



Initial Memory Allocations

Memory Allocations After Enabling The MMPU

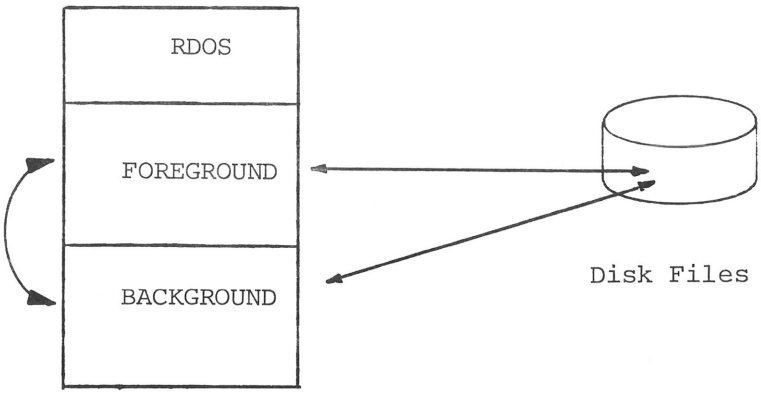
EXTENDING USER ADDRESS SPACE

Mapped addressing extends the total amount of resident memory, yet it does not itself permit any single user program to exceed 32K words of memory. Since this restriction is unacceptable to some application programs, RDOS provides two facilities to raise the total user program space to up to one hundred twenty-two K word blocks of memory. The actual amount of user memory depends upon the size of the core-resident portion of the operating system.

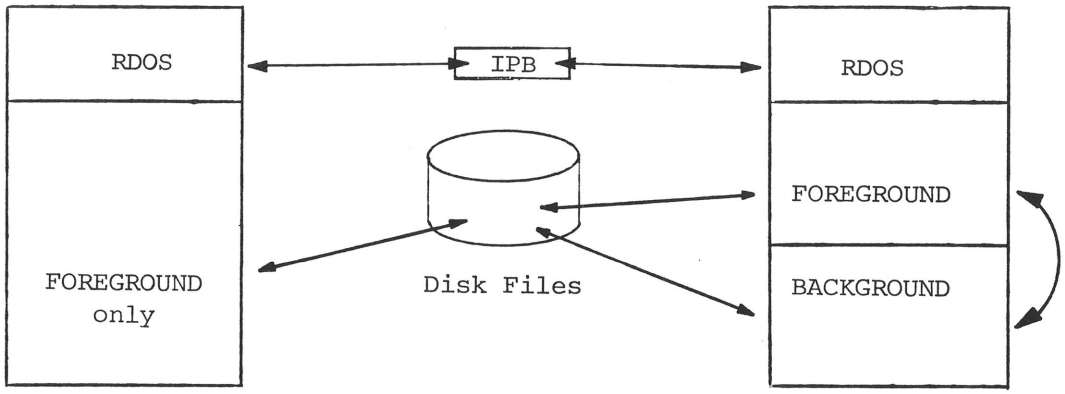
Since directly (or indirectly) addressable memory, even with mapped address space, cannot exceed 32K, MRDOS permits programs to access resident memory beyond this limit by means of extended address space. The two means provided by MRDOS for utilizing extended address space are virtual user overlays and window mapping. Both virtual user overlays and window mapping create extended address space by storing data into memory blocks outside the 32K address space directly accessible by the user. When this data or program material is to be accessed, the desired blocks are remapped into the user's address space by enabling the memory management unit. It is in the precise methods of storing the data and enabling the memory management unit that these two methods differ.

A virtual overlay is a section of disk data, specially identified at relocatable load time, which is loaded into extended address space when the overlay file is opened. This overlay is then made accessible by means of the same system or task call that is used to load an ordinary user overlay from disk. This use of an overlay load command merely triggers the memory management unit, as contrasted with an ordinary overlay load which requires a disk access and actual transfer of data.

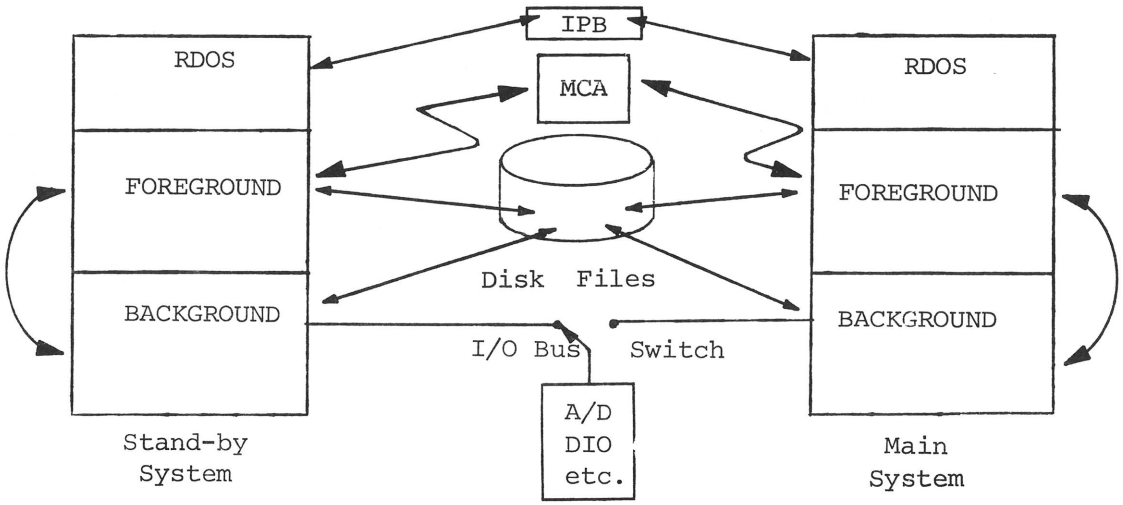
Window mapping provides a means of explicitly accessing the memory management unit. Data is mapped in 1K blocks to and from a logically fixed area in user address space, called a window, and extended memory. Windows, like virtual overlays, are defined in integer multiples of 1024-word blocks and are block aligned. Another similarity between window mapping and virtual overlay loading is that no true data transfer occurs. It is the addresses at which the data is found that are changed. This accounts for the high speed at which data in extended memory can be accessed.



Single Processor / Foreground-Background System



Dual Processor / Shared Disk System



Parallel Main and Standby System

DUAL-PROCESSOR AND MULTI-PROCESSOR SHARED-DISK SYSTEMS

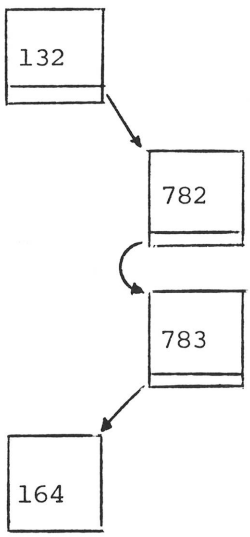
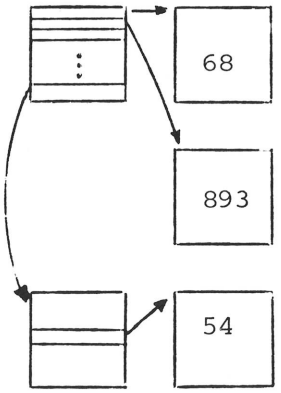
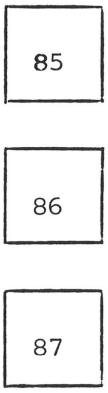
As the cost of small computers continues to decline, many new application areas are developing for multi-processor systems. Such systems commonly consist of two CPUs which share some or all of the system peripherals, thus lowering the total cost of the system. More than two processors can also be interconnected, however, to increase the flexibility and power of an over-all system.

The single processor system can have both a foreground and a background program, each working independently of the other. These two programs, however, can communicate via common disk files and via certain system and task calls. A natural and inexpensive outgrowth of this arrangement is the dual processor system. The dual processor system may simply consist of 2 CPUs, each with a foreground and background, and each operating with a considerable degree of independence of the other. Backgrounds and foregrounds under each CPU in such a system retain the ability to talk to one another, and all four programs may communicate via disk files. An inter-processor buffer (IPB) is required whenever two processors access a common disk, so that truly simultaneous attempts to access the disk will be handled in an orderly manner.

In critical real time situations, redundancy may give a measure of safety to a total system, assuring a continued operation even when catastrophic failure occurs in a major system component. Such dual-processor systems contain a main system and a back-up system. The main system controls or monitors some process or series of processes continuously, and the back-up system stands ready to assume the main system functions in the event of failure in the main system. While in the standby state, the back-up system can be employed on lower priority tasks such as data analysis, summary reporting, and the development of new real time programs. Both the main and the back-up systems can run under RDOS, either in a foreground/background arrangement or as a single real time program.

If the main system fails, some method for detecting the trouble must be available. Such methods include software cross checking of critical functions and the use of such special hardware as a watchdog interval timer (incorporated in the IPB hardware). If failure occurs, transfer to the back-up system must be made quickly and smoothly. Process information can be passed via disk files, tape files, or via a multiprocessor communications adapter (MCA). This last device permits full-duplex asynchronous communications to occur between two or more processors. In all dual processor systems, some method of communicating between the two processors must be maintained continuously.

Finally, multiprocessor systems can be built under RDOS where more than two CPUs are connected, perhaps in a master-slaves arrangement. Each smart slave in such a system could have either unique or redundant tasks, and all would be managed by a supervisory CPU. All CPUs would run under RDOS in either a background only or foreground/background environment, and each CPU could talk to others via an MCA line.

	Sequential	Random	Contiguous
TYPE:			
CREATION:	.SYSTM .CREATE	.SYSTM .CRAND	.SYSTM .CCONT
EXTENDIBILITY:	yes	yes	no
ORGANIZATION:			
ACCESS COMMANDS:	RDL/WRL RDS/WRS RDR/WRR	RDL/WRL RDS/WRS RDR/WRR RDB/WRB	RDL/WRL RDS/WRS RDR/WRR RDB/WRB
MAXIMUM REQUIRED DISK ACCESSES:	(W/255.)+1 where W is the desired word number and integer division is used	((W/(255.*266.))+1)+1 where W is the desired word number and integer division is used	1

DISK FILE STRUCTURES

Sequentially Organized Files

Sequential organization is the simplest organization to understand. Information in sequentially organized files is stored in groups of disk blocks. The last word of each 256 word block is used to store a pointer to the next block in the file. This pointer is invisible to the user, and is solely for system use. Each 256 word block has a unique address called a logical disk address which is derived from the physical disk sector/track addresses. Distinct from the logical disk address is the logical or relative block number which denotes the relative position of a block of data within its disk file.

The logical disk addresses of a sequentially organized file need not be (and seldom are) in an unbroken series. When building a sequential file the system simply appropriates the next available disk block when storage is needed, and constructs a pointer to the block. Sequentially ordered blocks are sequential in this sense: After processing any given block, the system may step either to the previous block or to the next block in the series. To access the tenth block after the first block, the system would have to read the nine intervening blocks -- a time-consuming process.

Randomly Organized Files

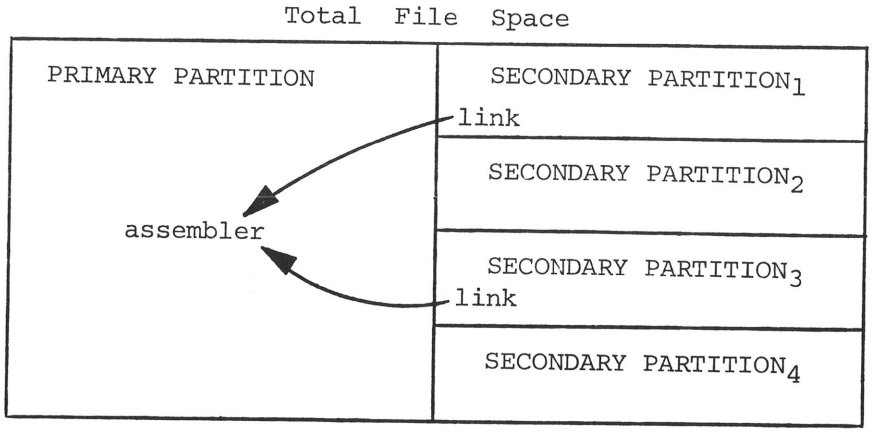
Random file organization provides the best combination of flexibility and accessibility of data. In randomly organized files a master index of all logical disk addresses is created. The master index blocks themselves are sequentially organized files.

Blocks of data storage in random files utilize all 256 words for information storage. Each block is assigned a sequential positive integer by its position within the master index, indicating the block's logical position within the file. In processing randomly organized files, two disk accesses at most are all that is generally required for the reading or writing of each block: One to access the file index and one for the block of data itself. If the index is core resident (having previously been read into a system buffer), only one access need be made. In most cases the index will be core resident.

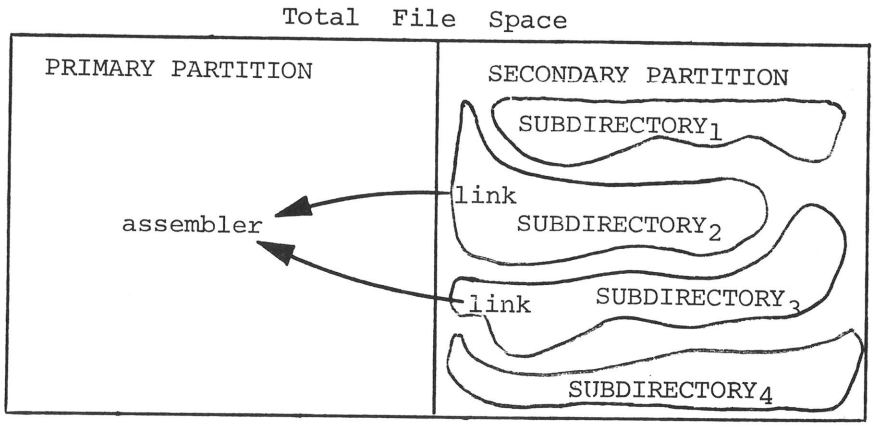
Contiguously Organized Files

Contiguous file organization has a rigid structure yet provides the quickest access to data. Contiguous files are composed of a fixed number of disk blocks which constitute an unbroken series of logical disk addresses. These files can neither be expanded nor reduced in size, since by definition they occupy a fixed series of disk blocks. Contiguous files may be considered as files whose blocks may be accessed randomly but without the need for a random file index.

All I/O operations which can be performed on randomly organized files can be performed on contiguously organized files. Contiguous files have the advantage of usually requiring less time for accessing blocks within the file. The drawback to contiguous files is that they may not always be created, and may never be changed in size. Their creation depends upon the availability of the required number of free neighboring disk blocks.



Secondary partitions guarantee a fixed amount of file space will be available,



while subdirectories use file space more flexibly.

Disk Partition Techniques

PROTECTING AND SHARING DISK FILES

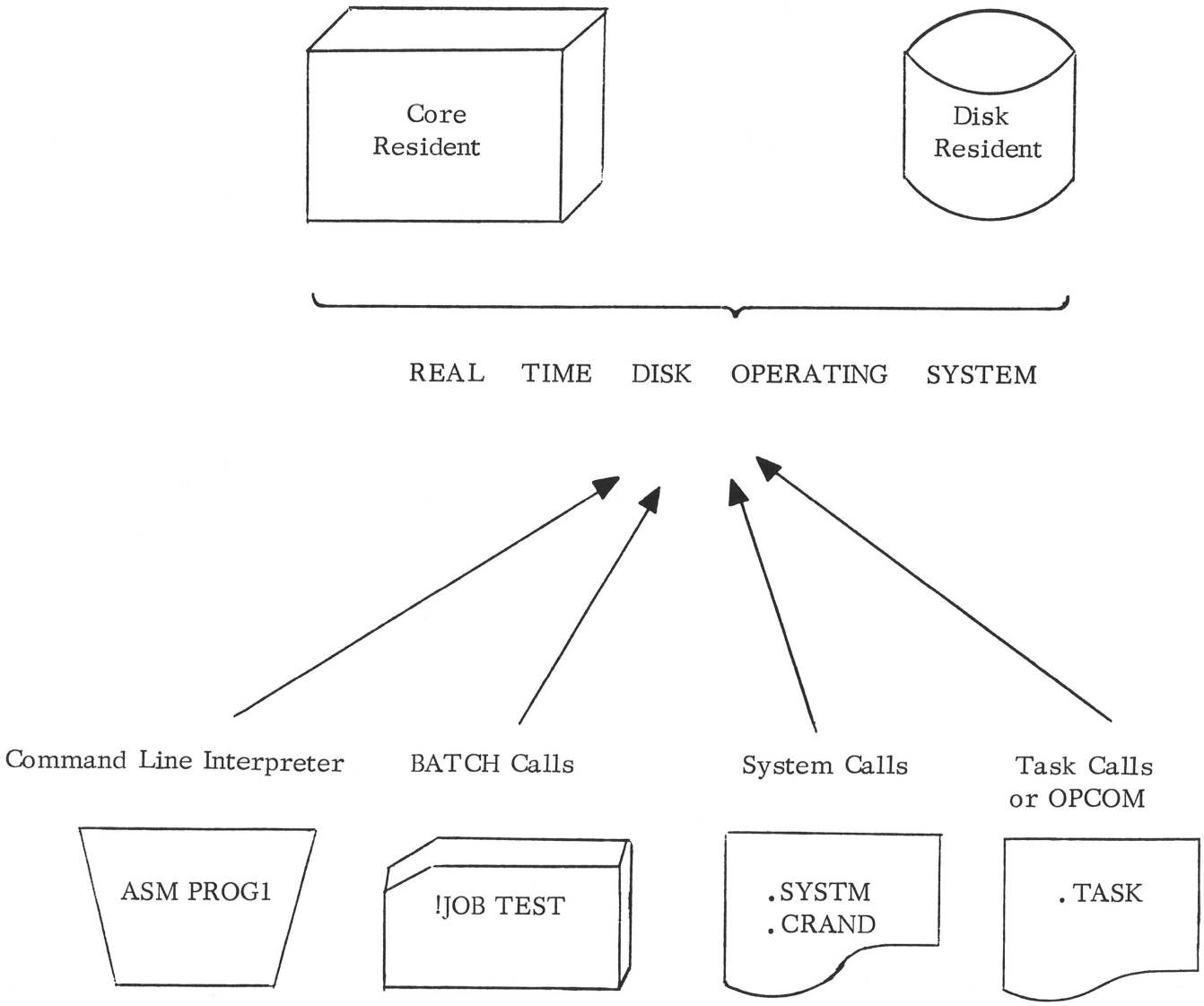
In both single and multi-processor systems there is often the need to protect certain files from being read by outsiders, permitting certain files to be read but not to be altered, and permitting copies of certain files to be shared for use by independent tasks. In response to these needs, Data General Corporation has implemented disk partitioning and the linking of disk files.

Before elaborating upon the concepts of partitioning and linking, we must consider the notion of file directories. A disk file directory is merely an index which lists the files in a portion of disk file space. A directory is similar to the index in a department store catalogue in that both the directory and the catalogue index describe where certain items can be found. The disk file directory, however, also contains other information about each entry over and obove its location and name, such as the file's attributes, the time the file was created, etc. A single unpartitioned disk has only one directory, but if the disk space is subdivided into smaller logical areas, each area has a directory which describes the files in that file space. Before any file can be opened and accessed,, its associated directory must be initialized.

A disk partition is simply a portion of total disk file space; each partition has a directory which describes and locates the files contained within the partition. Before disk file space is subdivided, the entire space is called a primary partition. One or more portions of this file space may be separated from the primary partition, and these separate portions are called secondary partitions. Secondary partitions are fixed in size at the time they are created. It is possible to create one or more subsets of either primary or secondary partition file space called subdirectories. A subdirectory is a portion of a parent file space and its file space may increase or decrease in size within the limits of the total file space of the parent. Subdirectories are mutually exclusive sub-sets of parent file space. Primary partitions, secondary partitions, and subdirectories are all often called simply directories.

The following illustration compares and contrasts the advantages of disk partitioning and subdirectories as a means of sharing disk space among 4 users. One of two disk allocation schemes might be used to protect each user's files from being read or altered by another user: 1) create 4 secondary partitions, and 2) create a single secondary partition (4 times as large as in 1) above) and assign each user to a subdirectory within this partition. Both options allow each user's disk files to be protected. The first plan guarantees a fixed amount of file space to each user, but limits file space to this fixed amount. The second plan allows each user to grab and later release as much file space as he requires from the secondary partition as long as there is any unused space.

Finally, RDOS also permits the referencing of a file by one directory which exists in the file space described by the same or another directory. This mechanism, called a link entry, permits a resolution file to be pointed to and accessed from within a directory which may or may not describe the file space containing the file. In their most common application, link entries permit the conservation of disk file space by allowing a single copy of a commonly used disk file (like the FORTRAN compiler, assembler, etc.) to be employed by users in all directories. Resolution files may be specified to be non-linkable, non-writable, etc., at the discretion of the owner of the resolution directory.



Communicating with RDOS

COMMUNICATING WITH RDOS

There are four principal ways for a user to communicate with RDOS. These ways are: (1) via the Command Line Interpreter (CLI), (2) via the BATCH monitor, (3) with system calls, and (4) with task calls.

System calls and task calls are issued as program instructions which are assembled and then are relocatably loaded before they can be activated. System and task calls activate logic within either the system or task modules. One task module, the Task-Operator Communications processor (OPCOM), permits a user to sample and/or modify the state of tasks by means of commands issued from the console keyboard ($TTI).

The Command Line Interpreter (CLI) is a system program, executable in either the foreground or the background, that accepts command lines from the console and translates this input as commands to the operating system. The CLI acts as a console interface between the user and the system, and it permits user programs to be executed and utilities to be run, it performs certain file housekeeping chores, and it provides a diversity of other functions for the user acting at the console keyboard.

RDOS restores the CLI to main memory whenever the system is ready to execute a program--after initialization, after a disk bootstrap, after a console break, after the execution of a save file, etc. The CLI, in turn, indicates to the user that the system is ready to accept commands by typing a ready message on the console. The message consists of "R" followed by a carriage return.

The user activates CLI responses to a command by typing a line and pressing the RETURN key. The operation of the CLI may be saved or aborted by typing the CTRL key simultaneously with either the letters C, F, or A.

The BATCH monitor provides another means of focusing the power of RDOS onto user applications. BATCH permits a collection of user programming jobs to be performed one after the other, without the need for constant operator intervention. BATCH allows programs to be compiled, assembled, loaded, and executed by means of command words inserted by the operator into each of the jobs. BATCH also performs other functions appropriate to a serial job situation, and it also maintains a log of system usage. The CLI is used to activate BATCH, and--like the CLI--BATCH may be run in either the foreground or the background of a computer with memory mapping.

CLI Command Repertoire

Utility Calls

ALGOL	- Compile an ALGOL source file.
ASM	- Assemble a source file.
BASIC	- Execute a BASIC program.
BATCH	- Define a BATCH job stream.
CLG	- Compile, load and execute a FORTRAN IV program.
DEB	- Read a program and go to the debugger.
EDIT	- Perform a string edit.
FORT	- Compile and assemble a FORTRAN IV program.
FORTRAN	- Compile and assemble a FORTRAN 5 program.
LFE	- Update a library file.
MAC	- Perform a macro assembly.
NSPEED	- Perform an enhanced edit of a NOVA disk file.
OEDIT	- Edit a disk file.
OVLDR	- Perform an overlay load.
RLDR	- Perform a relocatable load.
SPEED	- Perform an enhanced edit of an ECLIPSE disk file.
SYSGEN	- Generate an operating system.

Directory-File Maintenance

APPEND	- Concatenate two or more files.
BPUNCH	- Punch a binary file.
BUILD	- Build a file of file names.
CCONT	- Create a contiguous file.
CDIR	- Create a subdirectory.
CHATR	- Change a file's attributes.
CHLAT	- Change link access attributes.
CLEAR	- Set file use counts to zero.
CPART	- Create a secondary partition.
CRAND	- Create a randomly organized file.
CREATE	- Create a sequential file.
DELETE	- Delete a file.
DIR	- Change the current directory.
DUMP	- Dump one or more files.
EQUIV	- Assign a new name to a directory specifier.
FILCOM	- Compare two files.
FPRINT	- Print on the line printer in hexadecimal and in ASCII the contents of a file.
GDIR	- Print the current directory name.
GSYS	- Get the current system name.
INIT	- Initialize a directory or device.
LINK	- Create a file link.
LIST	- List file directory information.
LOAD	- Reload dumped files.
MDIR	- Get current master directory's name.
MKABS	- Make an absolute file.
MKSAVE	- Make a save file.
MOVE	- Move a file between directories.
PRINT	- Print a file on the line printer.
PUNCH	- Punch a file.
RELEASE	- Release a directory/device.
RENAME	- Rename a file.
REPLACE	- Perform an overlay file replacement.
REV	- Determine a current revision level.
SAVE	- Save a core image.
SQUASH	- Prepare a system save file for bootstrapping.
TPRINT	- Print the contents of the tuning file.
TYPE	- Type a file on the console.
UNLINK	- Delete a link.
XFER	- Copy the contents of a file to another file.

Foreground-Background Calls

EXFG	- Execute in the foreground.
GMEM	- Get the foreground/background mapped memory sizes.
SMEM	- Set the foreground/background mapped memory sizes.

Miscellaneous

BOOT	- Perform a disk bootstrap.
DISK	- List the number of disk blocks used and remaining.
ENDLOG	- Stop recording console output.
GTOD	- Get the time and date.
LOG	- Start recording console output.
POP	- Return to the next higher program level.
SDAY	- Set today's date.
SPDIS	- Disable spooling on a device.
SPEBL	- Enable spooling on a device.
SPKILL	- Stop a spool operation.
STOD	- Set the current time.

COMMAND LINE INTERPRETER (CLI)

The CLI can perform a variety of necessary functions for the operator sitting at the console terminal.

The CLI itself executes certain system commands such as CREATE and RENAME. More complex commands cause the CLI to build a file containing an edited version of the command line and then to load the program named in the command line for execution. When execution is complete, control is returned to the CLI.

Some of the functions performed by the CLI are:

- System Initialization and Installation
- File Creation and Maintenance
- Directory Creation and Maintenance
- Setting/Reading the Time of Day
- Spooling Control
- Interfacing to System Utilities

CLI commands can be stacked (that is, new ones can be issued for execution while the current command is still being performed), and a variety of symbol conventions can be added to the basic commands to extend their meaning. Among these symbols are global switches which are appended to CLI commands themselves and local switches which are appended to CLI command arguments.

A list of all CLI commands is given on the previous page. These commands range from the simple to the complex. A simple command,

```
DISK )
```

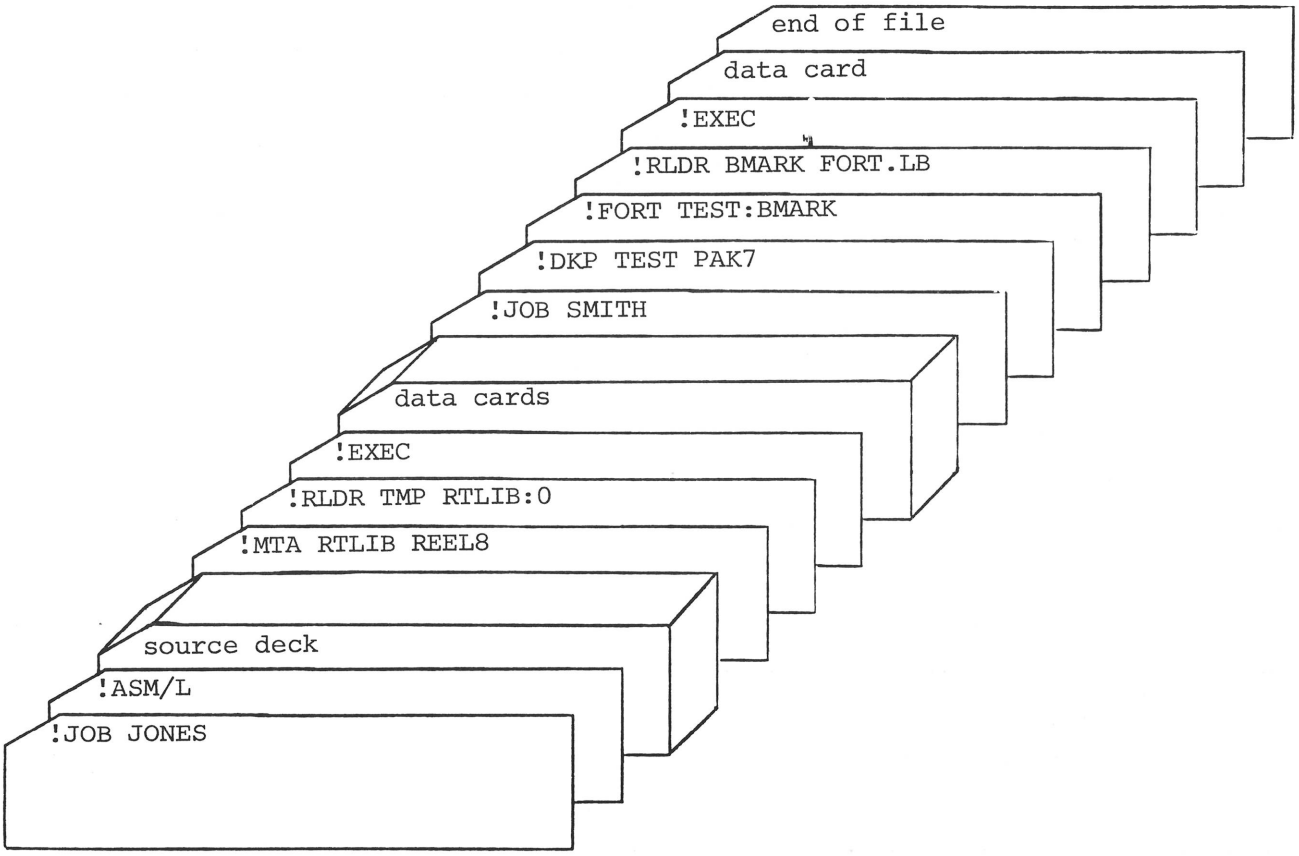
causes a count to be printed of disk blocks used and disk blocks still available on the current directory. A complex command like

```
RLDR/D  M1  M2  [01,02,03,04]  [P1,P2 P3,P4,P5]  $LPT/L )
```

causes a program complete with two overlay areas and the debugger to be loaded, with a copy of the load map printed on the line printer.

If commands issued to the CLI cannot be executed, the CLI responds with appropriate error messages. Commands requiring operator intervention (like mounting paper tape in a tape reader) cause prompting messages to be issued. The spooling feature (which will be discussed later) permits the CLI to execute new commands while it is waiting for previous I/O operations to be completed. Thus, program editing could be continued while an assembly listing is still being output.

To allow a user to take full advantage of the CLI and its command repertoire, and to permit the operation of the CLI within a 16K core configuration, much of the CLI has been subdivided into smaller, disk resident portions, which are called by the core-resident portion of the CLI as they are required.



BATCH Card Job Stream

USING THE BATCH MONITOR

The Real Time Disk Operating System contains a monitor called BATCH which permits a collection of programming jobs to be executed, one after the other, without the need for constant operator intervention. This collection of jobs may be input to the system via one or more job streams; each stream consists of one or more jobs input via a single device or disk file.

All BATCH commands have a clearly defined syntax and command line structure:

!command /modifier(s) arguments

The !JOB command, for example, indicates the start of a new job and assigns a name to the job, if one is given as an argument.

BATCH has many features which facilitate the running of jobs collectively. First, BATCH permits user jobs to reference symbolic devices, deferring actual device assignments until job execution time. This allows a program to refer to a magnetic tape transport as TAPE, for example, and allow the tape file references be resolved to MT0, MT1, MT2, MT17 etc. depending which transport is available at the time the program is run.

Secondly, BATCH permits jobs to be input via several streams, including disk files. This boosts operator efficiency, since one or more devices may be reloaded with jobs while other devices are still being used by the system. Clear system messages are output to the operator by BATCH, indicating when jobs should be mounted or dismounted from devices.

BATCH utilizes spooling, the outputting of data to slow output devices via a disk buffer. This gives the system independence from slow peripherals and allows more time to be spent on processing and executing jobs.

Finally, BATCH provides an accounting of system usage by all jobs, so that costs can be allocated accurately and system usage estimates can be obtained with ease.

Without getting into detail, the commands illustrated on the preceding page cause the following BATCH operations to occur. Two jobs are created, JONES and SMITH. JONES assembles a program, outputs the listing to the current listing device (SYSOUT), assigns the logical name RTLIB to a magnetic tape transport and identifies the needed reel as REEL8. The assembled program is then loaded and executed. SMITH, the second job, assigns a logical name TEST to the moving head disk (which has the identifier PAK7). FORTRAN program BMARK, on TEST, is then compiled, loaded and executed.

SYSTEM COMMAND LIST

Clock/Calendar Commands

.DELAY (delay the execution of a task), .GDAY (get today's date), .GHRZ (get the RTC frequency), .GTOD (get the current time), .SDAY (set the calendar), .STOD (set the clock)

User Defined Interrupt Servicing Commands

.DUCLK (define a user clock), .IDEF (identify a user device), .IRMV (remove a user device), .RUCLK (remove a user clock), .STMAP (set the data channel map)

Directory Maintenance and Control Commands

.CDIR (create a subdirectory), .CHSTS (get directory information for a file specified by channel number), .CPART (create a secondary partition), .DIR (change the default directory), .EQIV (assign a temporary name to a global specifier), .GDIR (get the current directory name), .GSYS (get the current system name), .INIT (open a directory/device), .MDIR (get the master device logical name), .RLSE (release a directory/device), .RSTAT (obtain resolution file directory information), .STAT (obtain file directory information)

File Maintenance and Control Commands

.APPEND (open a file for appending), .CCONT (create a contiguous file), .CHATR (change file attributes), .CHLAT (change link access attributes), .CRAND (create a random file), .CREATE (create a sequential file), .DELETE (delete a file), .GPOS (get the current file pointer), .GTATR (get file attributes), .LINK (create a link entry), .RENAME (rename a file), .SPOS (set the current file pointer), .ULNK (delete a link entry), .UPDAT (update file size information)

File I/O Commands

.CLOSE (close a file), .EOPEN (open a file for one user only), .ERDB (read an extended block), .EWRB (write an extended block) .GCHN (get a free channel number), .MTDIO (free format tape I/O), .MTOPD (open for free format tape I/O), .OPEN (open a file for one or more users), .RDB (read a block), .RDL (read a line), .RDR (read a random record), .RDS (read sequentially), .RESET (close all files), .ROPEN (open a file for reading only), .WRB (write a block), .WRL (write a line), .WRR (write a random record), .WRS (write sequentially)

Console I/O Commands

.GCHAR (get a console character), .GCIN (get the input console name), .GCOUT (get the output console name), .INTAD (provide keyboard interrupt state storage), .ODIS (disable console interrupts), .OEBL (enable keyboard interrupts), .PCHAR (output a console character), .RDOP (read an operator message), .WROP (write an operator message)

Overlay, Swap and Memory Commands

.BOOT (bootstrap a new system), .BREAK (save current memory as a save file), .ERTN (return from swap with error status), .EXEC (load and execute a swap), .MAPDF (define a window map) .MEM (determine available memory), .MEMI (allocate an increment of memory), .OVLOD (load a user overlay), .OVOPN (open an overlay file), .OVRP (replace an overlay file), .RTN (ordinary return from program swap), .WREBL (enable mapped blocks to be modified) .WRPR (write protect mapped blocks), .VMEM (size memory for window mapping)

System and Spooling Commands

.DDIS (prevent a mapped user from accessing a device), .DEBL (enable mapped user access to a device), .GMCA (get the current CPU number), .RDSW (read the console switches), .SPDA (disable spooling), .SPEA (enable spooling), .SPKL (kill spooling), .TUOFF (tuning reports off), .TUON (tuning reports on)

Foreground/Background Commands

.EXBG (checkpoint a mapped background program), .EXFG (execute a foreground program), .FGND (see if a foreground program is running), .ICMN (define an inter-program communications area), .RDCM (read a message from the other program), .WRCM (write a message to the other program)

SYSTEM CALLS

Another way that users can communicate with the Real Time Disk Operating System is by making a system call within a source program. The general form of all system calls is as follows:

```
.SYSTM
command mnemonic
error return
normal return
```

The mnemonic .SYSTM must precede each system command word. This mnemonic is defined as JSR @17, and causes control to be passed through the task scheduler to the system call processor, a core-resident portion of RDOS. The task monitor saves the task environment in the calling task's TCB and saves the contents of location 16 (=USP) before passing control to the call processor. A complete list of system command word mnemonics is given on the opposite page.

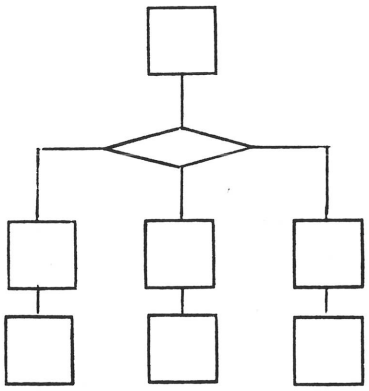
There are two basic command word formats:

1. command mnemonic

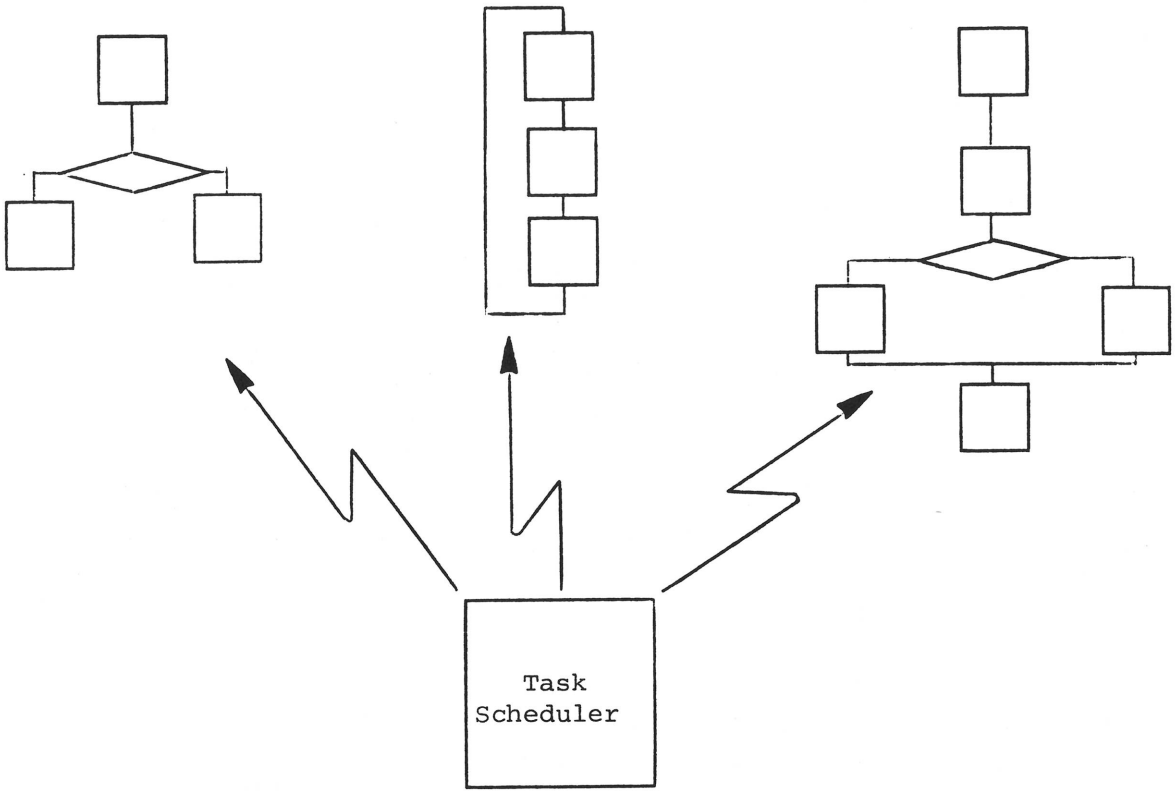
2. command mnemonic n

In the first format, the mnemonic is not followed by the integer n. In the second format, n represents a non-negative integer whose largest value may not exceed 77_8. Any system command requiring a channel number n need not specify this number explicitly in the command itself. By specifying n to be 77_8 the system will use the number passed in AC2 as the channel number. This gives the user a flexible runtime device selection method. If other arguments are required by the call, these are passed in accumulators. Since on NOVA systems and on ECLIPSE systems whose programs are loaded with TMIN the contents of USP are placed in AC3 upon the completion of each system call, this location can be used to save AC2 before the call is issued. On ECLIPSE systems, single task programs loaded with BTMIN and all multitask programs load the frame pointer, FP, into AC3 upon return. The contents of USP are restored to their former state (before the call) in all systems.

The error return is taken if some impediment is encountered in the execution of the call. If the error return is taken, AC2 will contain a code describing the error condition. Upon the successful execution of the call however, the normal return is taken.



Single Task Environment



Multi-task Environment

TASKS

A task is a logically complete, asynchronous execution path through a program, subprogram, or overlay which demands use of system resources (usually CPU control). Many tasks may be directed to operate in a single reentrant path, and each of these tasks may be assigned a unique priority. Single task environments are already familiar to users of non-real time operating systems. FORTRAN IV programs, most user assembly level programs, even system utility programs like the assembler, editor, etc., are all examples of programs running in a single task environment. A single task environment is a program which has a single unified path connecting all its program logic, no matter how complex the logic branches.

There is no compelling reason why a program should be limited to performing just one task, however. Indeed, most user environments contain a multitude of unrelated tasks which must be performed. It was the problem of efficiently controlling real time environments which led to the notion of multi-task real time operating systems.

One real time program may have from several to a virtually unlimited number of logically distinct tasks. Each task performs a specified function asynchronously and in real time (i.e., as close to instantly as possible). CPU control is allocated by the RDOS Task Scheduler to the highest priority task that is ready to perform or continue performing its function.

Examples of multitask environments are airline reservation systems, inventory control systems, process control operations, and communications networks with message queuing and switching. The individual tasks within an inventory control system might be the immediate updating of inventory totals from data received by remote terminals, and the reordering of those items whose quantities fall below specified reorder points.

Other examples of applications which can be performed as single tasks, yet whose efficiency is raised when they employ multitasking concepts, are the BASIC interpreter and the symbolic text editor. Original versions of these programs allowed only one user to be serviced for the duration of his job. Much idle processor time resulted, however. Multi-user BASIC and a multi-terminal symbolic text editor are now available from DGC, neither of which causes a noticeable lowering of system response to keyboard input and both of which raise the system's throughput.

TASK STATES

EXECUTING - A task has control of the Central Processing Unit.

READY - A task is available for execution, but it cannot be raised to the execution state until it becomes the highest priority ready task and the currently executing task is reduced to some other state.

SUSPENDED - A task is awaiting the occurrence or completion of some system or task call, or it is awaiting a specific real-time event.

DORMANT - A task has not been initiated in the system, or its execution has been terminated.

TASK STATES AND PRIORITIES

User multitask programs run under RDOS will have one task (called the "default task") initiated for them by default. If the user wants more tasks, he initiates one or more of these tasks by issuing the appropriate task call from this first task.

When a multitask environment is established, there becomes a requirement for a task scheduler to decide which task should be in the execution state. To enable the scheduler to function, each task is assigned a priority at the time of initiation (this priority can later be changed). There can be 256 levels of task priority in the range 0 to 255, with priority 0 being the highest. Several ready tasks may exist at the same priority level, yet in this case the tasks all take turns at being the task within the level which is capable of receiving control from the task scheduler.

The default task is initiated at the highest priority and since it is the only task in the system, it receives control. When this task initiates the second or subsequent tasks, the task scheduler is called upon to put the highest priority task into execution. Other tasks that have been initiated but are of lower priority are said to be ready to run. The task scheduler always gives CPU control to the highest priority task that is ready. Many tasks may be ready, but only the highest priority task receives control.

The executing task becomes suspended when it makes any call to the operating system. The task remains suspended until the call is completed, at which time it becomes readied again. Certain task calls can also cause an executing task to become suspended.

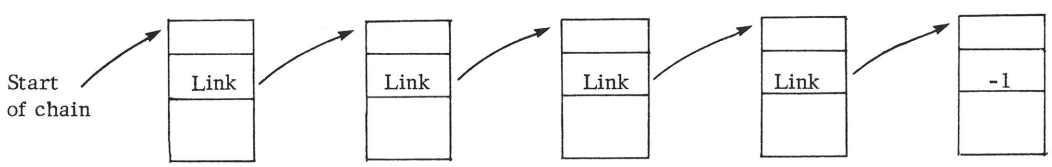
Actually there are two different types of task suspension and a task may become doubly suspended (i.e., unable to become readied until both reasons for its suspension are resolved). One type of suspension is caused by a task's issuing either a system call or a .XMTW/.REC task call (this will be elaborated upon later), or by requesting an occupied overlay area. Another type of suspension occurs when a task is suspended by one of the three task calls which perform this function specifically: .SUSP, .ASUSP, or .TIDS. No task may be raised to the ready state until both types of suspension are removed.

In sum, tasks can exist in any of four states. A task may be in control of the CPU and executing its assigned path, it may be ready and awaiting its turn to gain control, it may be suspended and unable to compete for control until it becomes readied, or it may be dormant since it has not yet been initiated into one of the other states.

Task Control Block Structure

word 0	User PC and Carry	The task's Program Counter.
1	AC0	The task's AC0.
2	AC1	The task's AC1.
3	AC2	The task's AC2.
4	AC3	The task's AC3.
5	Status bits and Priority	Task priority and task status.
6	System call word	Used by RDOS when task issues a system call.
7	Link	Pointer to the next TCB in the chain.
10	USP	Contents of location 16, used for general purpose storage or for FORTRAN STACK POINTER.
11	TELN	Pointer to the Run Time Stack segment associated with this task if it is a FORTRAN task. (Otherwise, unused.)
12	TID	Task identification number.
13	Temporary	RDOS temporary storage
14	Task kill address	Address of a special address to receive control when a task kill is attempted.
15	Stack pointer	Stack state save information (reserved for compatibility in single task NOVA environments and used in ECLIPSE environments).
16	Frame pointer	
17	Stack limit	
20	Overflow address	

TCB Chain



TASK ENVIRONMENTS

As discussed in the previous section, a task within an RDOS system can either be dormant or active. If the task is dormant, the system does not know that it exists even though the code could be resident within the user address space. When a task is active (executing, ready or suspended state) certain status information must be maintained about each task to enable the scheduler to maintain the highest priority ready task in the executing state.

This status information about each active task is contained within an information structure called a Task Control Block (TCB). There is one TCB for each task in the active state (and no TCB for a dormant task). TCBs are used to store active register states and other priority and status information when the task exists in either the ready or suspended state. The TCB of the executing task is reserved for the task but the TCB remains unused by the system until the task loses control of the CPU. If the task becomes readied or suspended, its TCB is then used to store its status information. If, on the other hand, the rescheduling resulted from the task terminating its own execution, its TCB is placed into a pool of available TCBs.

The TCBs of ready, executing, and suspended tasks are linked together in an active chain. This chain is organized in order of decreasing task priority. Each TCB in the active chain is connected by its link word to the next TCB in the chain. Among equal priority tasks, a round-robin scheduling of system resources is performed. Whenever a task has its TCB entered in the active chain, the task is automatically assigned the lowest position within a priority level. The last TCB in the active chain has a link of -1.

Unused TCBs in the system are linked together to form an inactive chain of available TCBs. Except for the link words, these TCBs are unused until a task is initiated, at which time a TCB is removed from this chain and is placed on the active chain.

Source Level Task Calls

Task Initiation

.QTSK	-	Initiate a core-resident or overlay task by time of day.
.TASK	-	Initiate a task.

Task Execution Control

.ABORT	-	Terminate a task immediately.
.AKILL	-	Kill all tasks of a given priority.
.ARDY	-	Ready all tasks of a stated priority.
.ASUSP	-	Suspend all tasks of a stated priority.
.DQTSK	-	Dequeue a previously queued task.
.DRSCH	-	Disable the rescheduling of the task environment.
.ERSCH	-	Re-enable the rescheduling of the task environment.
.KILAD	-	Define a kill-processing address.
.KILL	-	Kill the caller.
.OVEX	-	Release an overlay and return to the caller.
.OVKIL	-	Decrement an overlay use count and kill the caller.
.OVREL	-	Decrement an overlay use count.
.PRI	-	Change the priority of the caller.
.REMAP	-	Trigger the memory management unit for a window remap.
.SUSP	-	Suspend the caller.
.TOVLD	-	Perform a multitask overlay load

Inter-task Communication/Synchronization

.IOPC	-	Initialize the task-operator communications package, OPCOM.
.IXMT	-	Transmit a message from a user interrupt routine.
.REC	-	Receive a message.
.TRDOP	-	Read a message from the console keyboard.
.TWROP	-	Write a message to the console printer or screen.
.XMT	-	Transmit a message
.XMTW	-	Transmit a message and await its receipt.

Task Control by Identification Number

.IDST	-	Get the status of a task specified by I. D. number.
.TIDK	-	Kill a task specified by I. D. number.
.TIDP	-	Change the priority of a task specified by I. D. number.
.TIDR	-	Ready a task specified by I. D. number.
.TIDS	-	Suspend a task specified by I. D. number.

Miscellaneous

.LEFD	-	Disable LEF mode.
.LEFE	-	Enable LEF mode.
.LEFS	-	Get LEF mode status.
.SMSK	-	Modify the current interrupt mask.
.UCEX	-	Exit from a user clock routine.
.UIEX	-	Exit from a user interrupt routine.
.UPEX	-	Exit from a user power fail routine.

Task-Operator Communication Calls

DEQ	-	Dequeue a previously queued task.
KIL	-	Kill a task specified by I. D. number.
PRI	-	Change the priority of a task specified by I. D. number.
QUE	-	Queue a task specified by I. D. number for periodic execution.
RDY	-	Ready a task specified by I. D. number.
RUN	-	Initiate a task specified by I. D. number.
SUS	-	Suspend a task specified by I. D. number.
TST	-	Display the status of a task specified by I. D. number.

TASK CALLS

The Real Time Disk Operating System offers a complete set of task calls for the monitoring and control of a real time environment. These task calls can be considered in two broad categories: calls issued at the source program level, and task commands issued via the console keyboard.

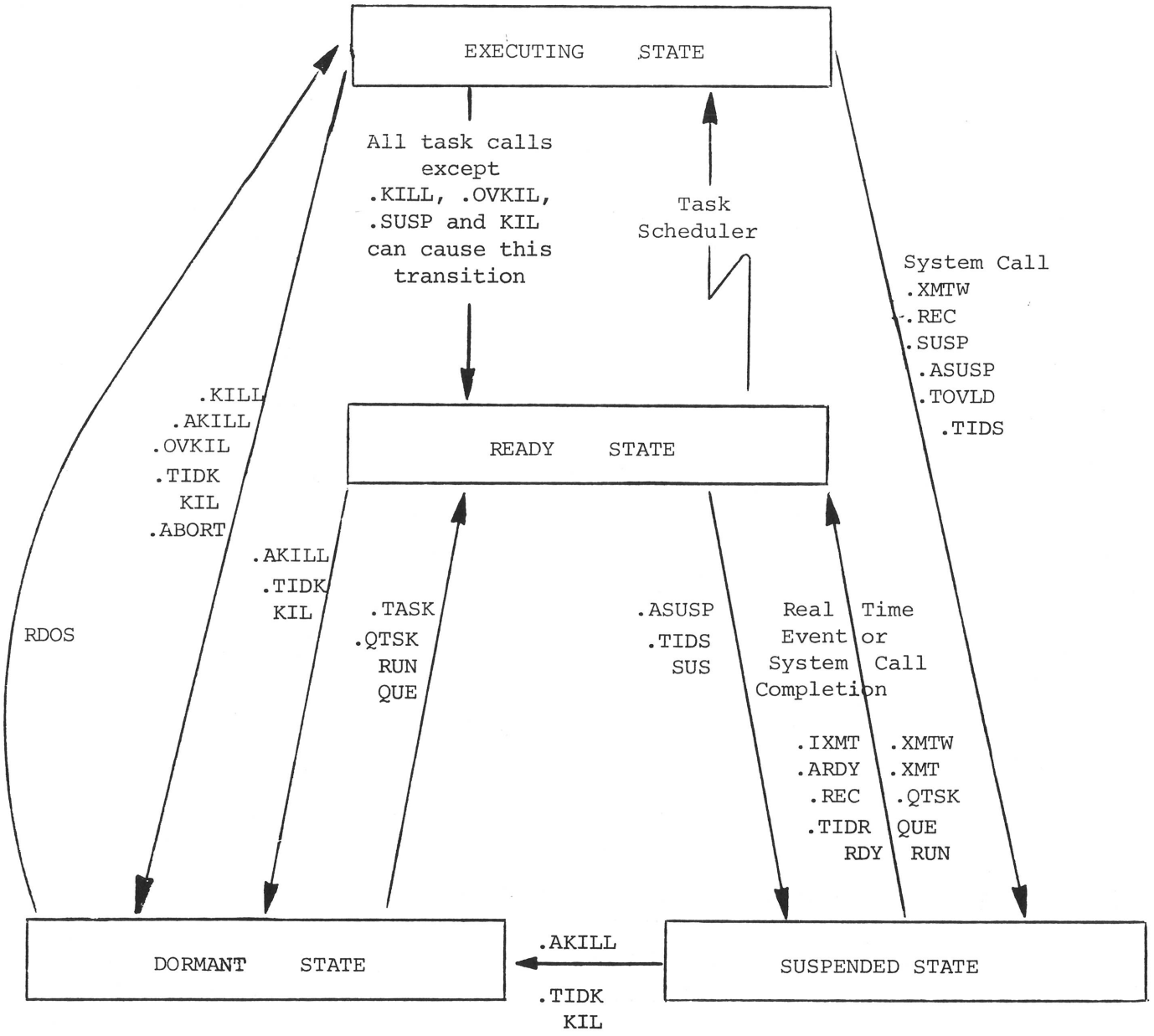
The source program is by far the most common origin of task calls, and all multitask programs must issue several calls from the source level. Source level calls can be considered in five distinct categories; 1) Task Initiation, 2) Task Execution Control, 3) Intertask Communication and Synchronization, 4) Task Control by Identification Number, and 5) User Clock and User Interrupt Control.

Before a multitask environment can be created, dormant tasks within the user program must be initiated, raised from the dormant to the ready state. Two task calls permit tasks to be initiated from the source level: .TASK and .QTSK. The default task is used to issue at least the first of these calls; after more than one task is running, any task can be used to issue these calls. Having initiated several tasks, RDOS provides a variety of task calls to give the user effective control over the tasks. Calls in the Task Execution Control category permits tasks to be suspended, readied, or killed as a priority class. Also included in this category are the task calls which permit overlays to be loaded and released in a multitask environment, and a call to trigger the memory management unit.

Effective control of a multitask environment requires that a communications facility exist between each of the tasks. Tasks are permitted to send messages to other tasks and to halt their own activity temporarily, if desired, until the transmitted message is accepted by the recipient task. RDOS also permits messages to be transmitted from user interrupt service routines, even though multitask activity is in suspension when user devices are receiving service. Finally, operator messages can be passed between tasks and the operator console. A special task call issued from the source level, .IOPC, enables the Task-Operator Communications package (OPCOM). OPCOM gives the console operator the capability to examine and influence running tasks from the keyboard directly. This class of commands gives a truly dynamic control over the multitask environment, and it bypasses the intermediate steps of source level adjustments and relocatable loading of a new user program.

RDOS also permits precise control to be exerted over individual tasks. The class of calls, Task Control by Identification Number, permits tasks to be specified by their individual, unique identification numbers.

The final category of source level task calls, User Clock and User Interrupt Task Control, applies multitasking concepts to non-standard user devices. Included in this category is a means of providing special power fail/restore service and the means of creating a logical real time clock which runs at a frequency that is a submultiple of that provided by the system clock.



Task State Transitions

TASK EXECUTION CONTROL

Unlike .SYSTM calls, task calls consist of single word calls which are used to initiate tasks, modify their priority, etc. Apart from their different function, the most significant difference between task and system calls is that system calls are executed in system space (memory occupied by RDOS), while task calls are executed in user program space. Thus except for small differences the size of RDOS space cannot be reduced by restricting the number and type of system calls issued within a user program. By contrast, the diversity of task call types issued within any user program directly affects the size of that program.

Each task call has an associated modular package of code in the system library (SYS.LB, to be discussed later) which is required to perform the call operation. This enhances core utilization since only those modules selected by the user on a source level will occupy program space at load time.

Each task call causes a task state transition to occur. Most task calls cause the issuing task to move from the executing state to the ready state. However, other task state transitions are possible. Task identification calls, for example, cause tasks that are identified by I.D. number instead of by priority to change from either the ready or suspended states to the dormant state, or from the suspended state to the ready state.

If the executing task issues a system call, and in some cases when it issues task calls .ASUSP, .REC, .SUSP, .TOVLD, or .XMTW, the executing task may be reduced to the suspended state.

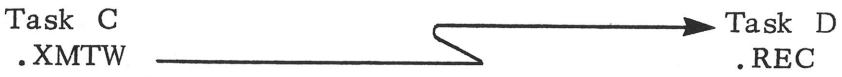
Issuance of other task calls can cause a suspended task to become readied. Calls in this category are .ARDY, .IXMT, .REC, and .XMT.

When the executing task issues a .TASK command, not only is the executing task lowered to the ready state but the task which has been activated is raised from the dormant state to the ready state. Conversely, whenever the task issues a .KILL call, the task issuing the call is reduced to the dormant state. .AKILL reduces all tasks of the specified priority level to the dormant state; this may include the caller, if his priority is the same as that specified in the call.

Finally, as described earlier, the Task-Operator Communications package (OPCOM) provides a user at the console with the facility to communicate with specific tasks while the program is running. OPCOM also allows certain task execution functions to be performed immediately, providing a rich variety of task execution controls. Using OPCOM contrasts with the practice of issuing task calls on a source program level.

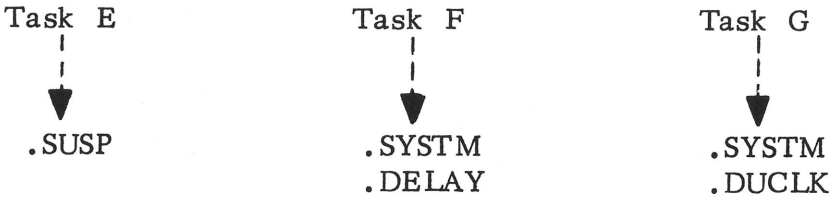
Task A sends message and goes into the ready state.
Task B is suspended until the message is received.

Intertask Communication



Neither task proceeds until message is passed from
Task C to Task D.

Task Synchronization



Task Timing Control

INTERTASK COMMUNICATION/SYNCHRONIZATION

Even though tasks operate asynchronously, it is often desirable for one task to be capable of communicating with another task. Tasks communicate with one another under RDOS by sending and receiving one word messages in agreed-upon memory locations. One word messages may, of course, be pointers to larger messages if the tasks agree beforehand on the use of such a technique.

A transmitting task may simply deposit the message in an agreed-upon location (by means of the .XMT call), or the caller may deposit the message and wait until its receipt (.XMTW). To receive such a message another task issues a .REC task call. If the transmitting task has not yet sent the message when the .REC call is issued, the receiving task waits until the message is sent. If the message has already been sent, the receiving task is readied; this indicates to the task that the requested message is now available. If the message was sent via a .XMTW task call, both the receiving task and the transmitting task are put into the ready state before control is sent to the task scheduler. One message may be transmitted to only one task.

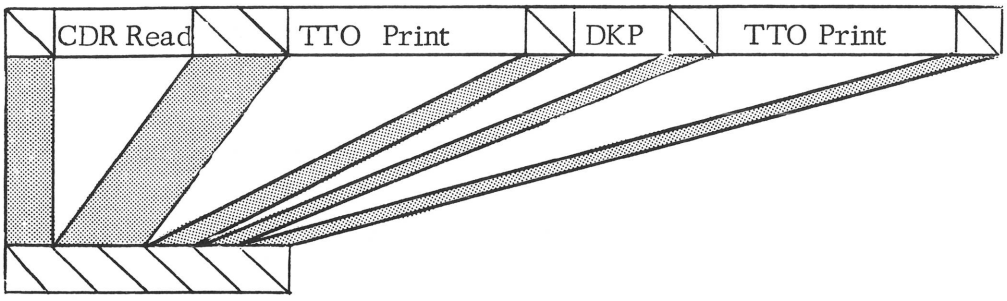
It is possible too for a message to be sent to a receiving task from a user interrupt service routine; this is done by means of the .IXMT call.

TASK TIMING CONTROL

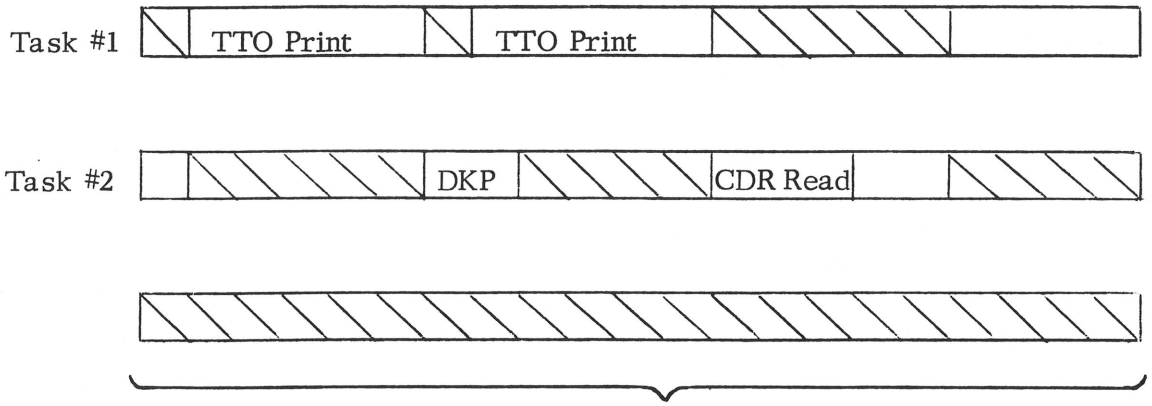
A task may suspend itself for a period of time which the task specifies. This allows users to implement a time slicing or round-robin allocation of CPU control to users. The delaying of a task is accomplished by means of a system call .DELAY. Tasks may suspend themselves for time periods that are multiples of the real time clock cycles.

RDOS also maintains a system clock and calendar for those tasks that should be scheduled on a time-of-day basis. Tasks may examine the frequency of the system clock and may obtain or set the date or the correct time in seconds, minutes, and hours. Moreover, core-resident and overlay tasks may be scheduled to gain control at periodic intervals by means of a call to .QTSK.

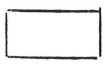
Finally, RDOS provides a pair of system commands which permit the definition and removal of a user clock driven by the system clock. This user clock generates interrupts at user-definable intervals. When one of these interrupts occurs, the Scheduler and task environment are frozen and control goes to a user-specified routine outside the multitask environment.



Single Task Operating System



Multiple Task Operating System



I/O Processing or Task Suspension

Useful System Processing

RDOS INPUT-OUTPUT CONTROL

An important function of any real time operating system is the efficient handling of input-output operations. Optimum usage of machine devices and central processor time in the accomplishment of tasks is the real reason for designing and implementing a multi-tasking system.

Since I/O devices are slow compared to the internal speed of the computer, they must be programmed to overlap their operations with computations, when possible, in order to:

- Increase usable CPU time by allowing one task to operate while I/O is in progress
- Greatly increase efficiency of I/O operations
- Provide more throughput of data by removing bottlenecks caused by slow peripherals (like Teletypes and plotters)

The responsibility of RDOS I/O control is to react during normal program execution to the structuring of I/O requests, making assignments of requests to machine devices when they are idle, and queuing requests for devices which are busy. Through the queuing facility, RDOS makes it possible to achieve maximum and continuous overlap of multi-tasks without direct intervention by the tasks themselves.

All input and output of data via devices permanently installed in a mapped system must be done via system I/O commands. Unmapped systems do not reject any user I/O commands, but the issuance of any such commands by a user would be both risky and unnecessary since a full complement of system I/O commands is provided.

As described earlier, system I/O commands require a channel number (0-76) to be given in the second field of the command word. This channel number is assigned to a particular device or file when the device or file is first opened. Devices are usually opened by means of a system command, e.g., .OPEN, which associates a given file name with a channel number. Having made this association, all commands pertaining to the file merely require that file's channel number.

INPUT/OUTPUT COMMANDS AND MODES

Type	Call	Data Type	Quantity
Single Character	.GCHAR/.PCHAR	ASCII	1 character
Line	.RDL/.WRL	ASCII	up to 132 character field terminated by a carriage return, form feed or null
Sequential	.RDS/.WRS	binary	field size controlled by byte count
Random Record Access	.RDR/.WRR	binary	64-word record
Direct Block	.RDB/.WRB	binary	logical 256-word blocks
Free Format	.MTDIO	binary	word count from 2-4096
Operator Message	.RDOP/.WROP .TRDOP/.TWROP	ASCII	field terminated by carriage return
Extended Direct Block	.ERDB/.EWRB	binary	logical 256-word blocks

INPUT/OUTPUT COMMAND MODES

The system provides eight basic modes for reading and writing data: Single Character, Line, Sequential, Random Record Access, Direct Block, Free Format, Operator Message, and Extended Direct Block.

In Line Mode, data read or written is assumed to consist of ASCII character strings terminated by carriage returns, form feeds, or nulls. Reading or writing continues until one of these three characters is detected. The system handles all device-dependent editing at the device driver level. For example, line feeds are ignored on paper tape input devices and are supplied after carriage returns on all paper tape output devices. Furthermore, neither reading nor writing requires byte counts, since reading continues until a terminator is detected and writing proceeds until a terminator is written.

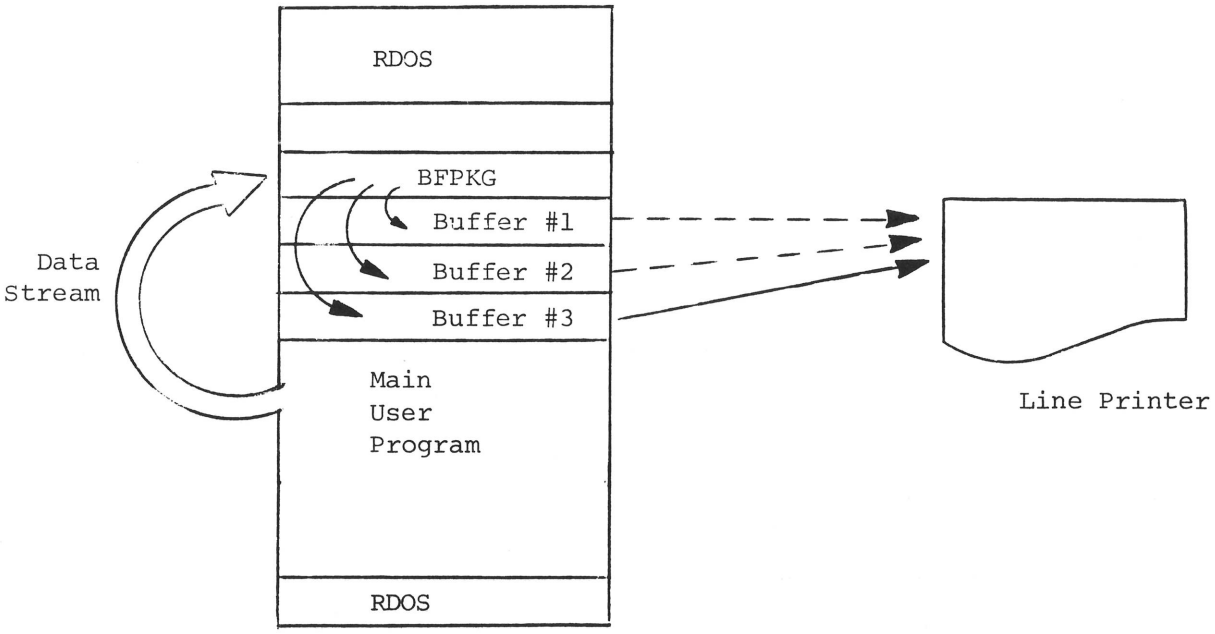
Sequential mode provides unedited data transfers. In this mode, no assumption is made by the system as to the nature of the information. Thus this mode would always be used for processing binary data and could also be used for processing ASCII data (provided no editing of this data is required). Sequential mode transfers require specific byte counts in order to satisfy read or write requests. All I/O devices may be used in sequential data transfers.

Random Record Access mode permits 64-word segments of disk block data to be accessed randomly. Data which is transferred in this mode may be either ASCII or binary, although no device level editing occurs.

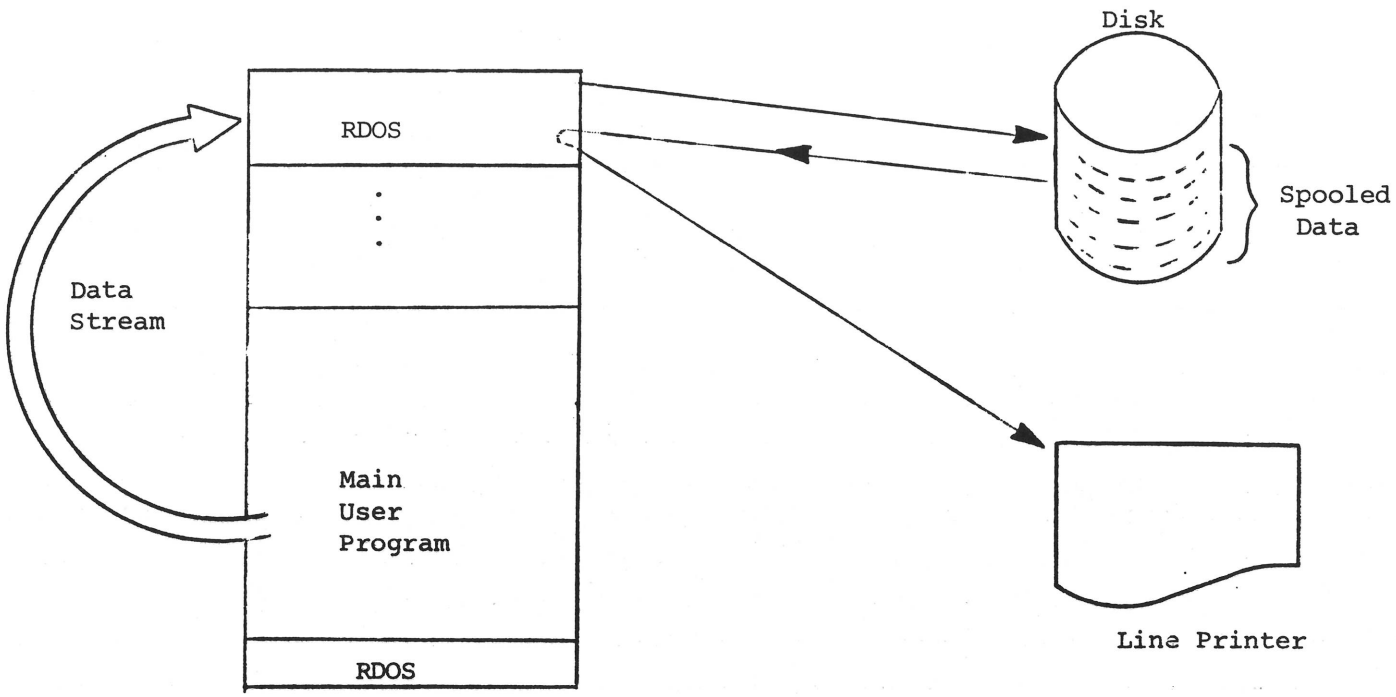
In Direct Block mode, binary data is transferred directly between a disk file and a specified memory area. This specified core area takes the place of the system buffer which is required for all other types of data transfers. The elimination of a system buffer makes direct block transfers faster than all other types of data transfers. Extended Direct Block mode, similar to Direct Block Mode, permits binary data to be transferred between a disk file and an extended memory area.

Free Format I/O permits the operation of magnetic tape and cassette units on a machine level: Reading and writing of data in variable length records from 2 to 4096 words within a record, spacing forward or backward from 1 to 4095 data records or to the start of a new file, and other similar machine level operations.

Two additional system commands are provided which permit unedited character transfers via the console teletypewriter: .GCHAR and .PCHAR. Two other system calls permit operator messages to be passed between the console and either the foreground or background: .RDOP/.WROP. Two task calls, .TRDOP/.TWROP, also give this facility.



Buffered I/O Package, BFPKG



Simultaneous Peripheral Operation On Line (SPOOLING)

SYSTEM INPUT/OUTPUT BUFFERING

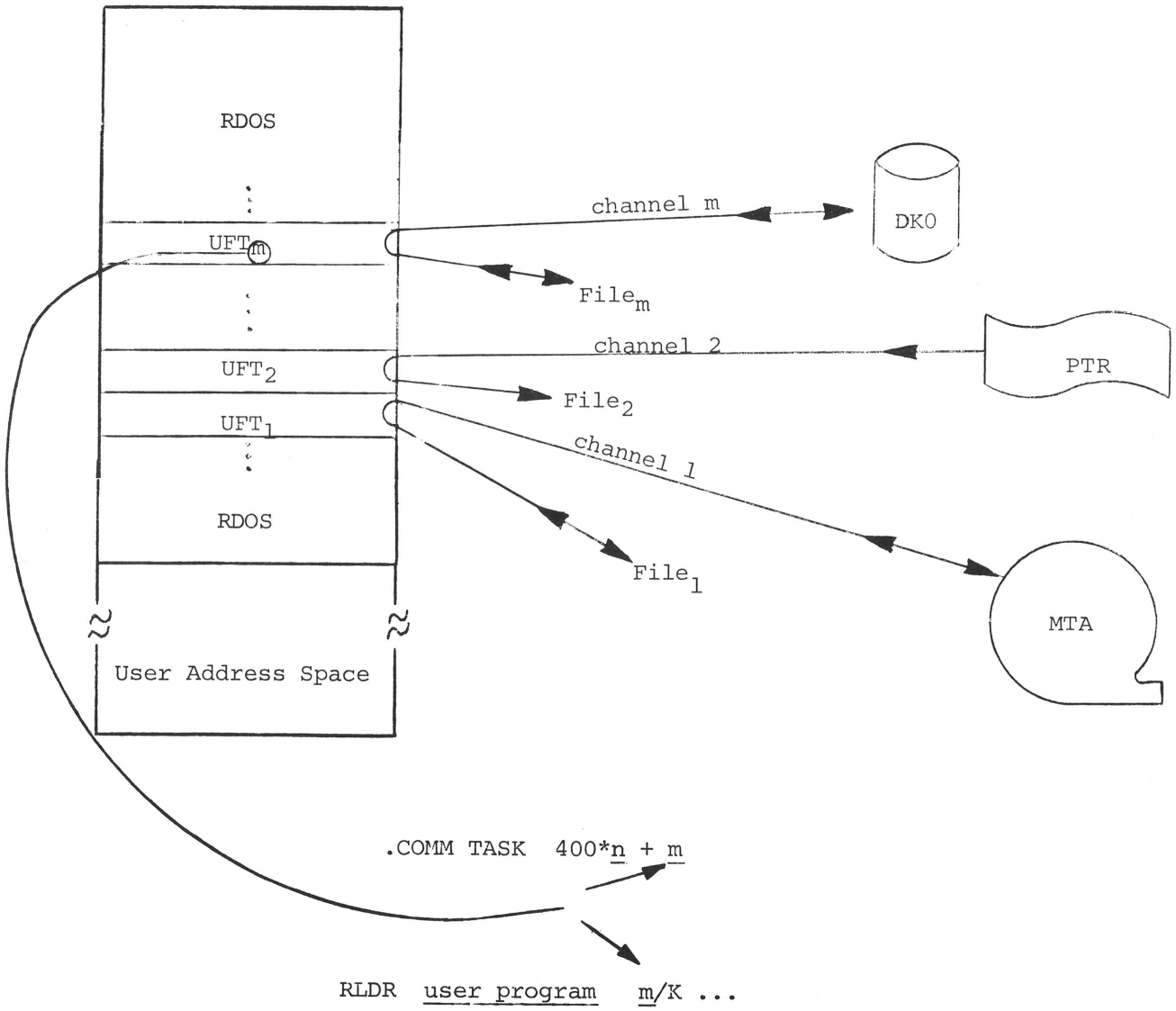
All data transfers (except Direct Block) to or from disk files and hardware devices are buffered in the operating system before the data is transferred into the user's buffer. Each system device handler has a small buffer associated with it depending on the speed of the device, e.g., 162 (decimal) words for the card reader, 40 bytes for the paper tape punch, and 160 bytes for the line printer. The other area is the system buffer area which is organized into blocks of 256 words each. When transferring data from a disk file it is first read into this buffer area before accessing the data within the block. This allows smaller transfers of data to occur to or from the user area, and frees the user from having to provide sector buffers in the program.

SPOOLING

Efficient I/O handling is the most important single factor in the effective utilization of CPU time. When messages are output on a slow device like a Teletype and its buffer fills up, the calling task will be suspended until the buffer is emptied. When spooling is provided, the next message called is temporarily stored on disk, and is later returned to core when the current message is completed. The significance of spooling is that queuing of output messages or information can now be accomplished easier without putting excessive loads on user core partitions. This also frees the user from having to optimize his message requests, thus permitting more effective use of the device. Spooling normally operates transparently on the Teletype, paper tape punch, plotter, and line printer but system and CLI commands allow the user to control spooling.

BUFFER CONTROL PACKAGE

The RDOS system library provides a module which permits faster line and sequential I/O transfers than is possible using the system I/O calls discussed previously. It utilizes tasking concepts to fill two or more buffers asynchronously and therefore provides a constant supply of data for program processing.



Establishing Channels to Devices and Files

DISK FILE PROTECTION AND CONTROL

RDOS provides several means to ensure disk file integrity and to provide the user of a file with control over the use and modification of his file. The primary means of data base protection is afforded by a series of tables found in system space on mapped RDOS systems; these tables are called User File Tables, UFTs. (In unmapped systems, UFTs reside at the top of free user space, above NMAX.)

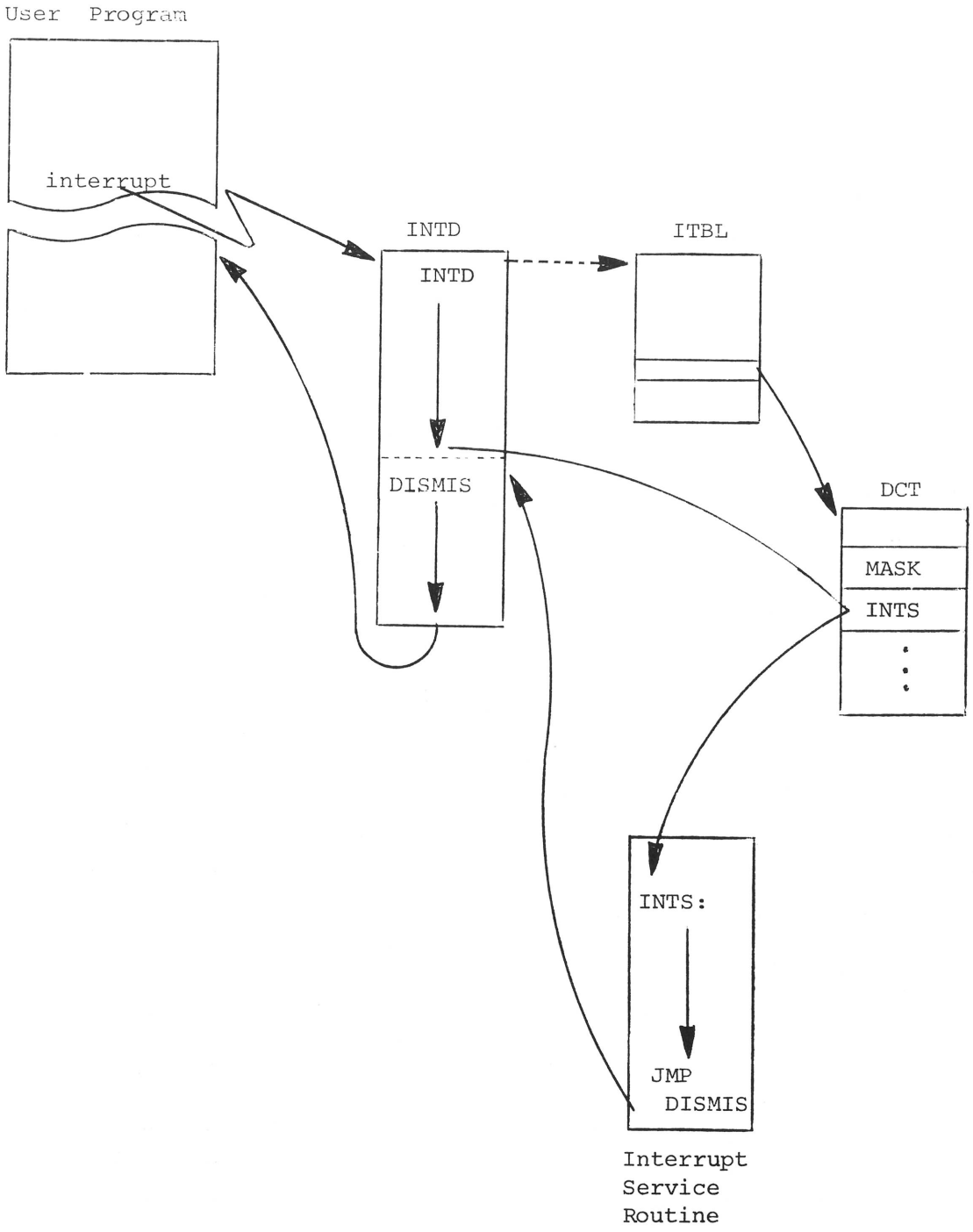
Since there are commonly more files and devices than there are communication channels to them, the UFTs provide a link between the channels and the devices or files to indicate which one is currently open on which channel. There is one UFT for each channel which will be used in either the foreground or the background. The number of channels which will be used by each save file is specified either on a source level by means of the .COMM TASK assembler pseudo-op or at relocatable load time via local switches. The maximum number of background and foreground channels which can be specified in a mapped system is defined at the time the operating system is generated.

All information required for channel-related processing of a file or device is maintained within the UFT. This information includes the file's name, its extension, its attributes, file access information, and the logical name of the device connected to the file via its channel. UFTs cannot be directly accessed by user program references, thus assuring the integrity of file and channel information. UFD frames can be examined, however, by means of either system call .STAT or .RSTAT ("get a file's status").

An additional measure of control is given to users whereby they may restrict the assessibility of files which they have opened. These means are the different ways that the files may be opened. The most common way of opening a file is via system call .OPEN. This provides a user with a non-exclusive capability to read and modify the contents of a file. That is, this type of open does not prevent the file from being opened by other users, nor does it forbid any users from modifying the file's contents at will. Another system open command, .ROPEN, also permits several users to open a file simultaneously (although on different channels), yet it prevents any of these users from modifying the file's contents. This open is a read-only open, and it prohibits the altering of the file's contents by any user who has issued this call.

RDOS provides a third system open command, .EOPEN, which is more restrictive than the other open commands. .EOPEN permits one exclusive modifier of a file for the period that he has it opened. This command gives a user special access rights to a file, permitting this user and him alone to both read and alter the contents of the file.

Lastly, as described earlier in the discussion of disk partitions and subdirectories, users in one directory may forbid users in other directories from resolving links to files in their directories. To accomplish this end, users simply define their files to be unresolvable by links. Other file attributes may be specified in the file's link access word to modify the accessibility of the file to link users.



Flow of Control During Interrupts

INTERRUPT SERVICING PROGRAM

When an interrupt is detected by the hardware, the currently executing program is suspended and control goes to an interrupt dispatch program, INTD. INTD is an integral portion of the RDOS system and resides in memory at all times. It directs control to the correct servicing routine by using the device code as an index into an interrupt branch table. The entry in this table is the address of a device control table (DCT) associated with the servicing routine.

The first three entries of the DCT are as follows:

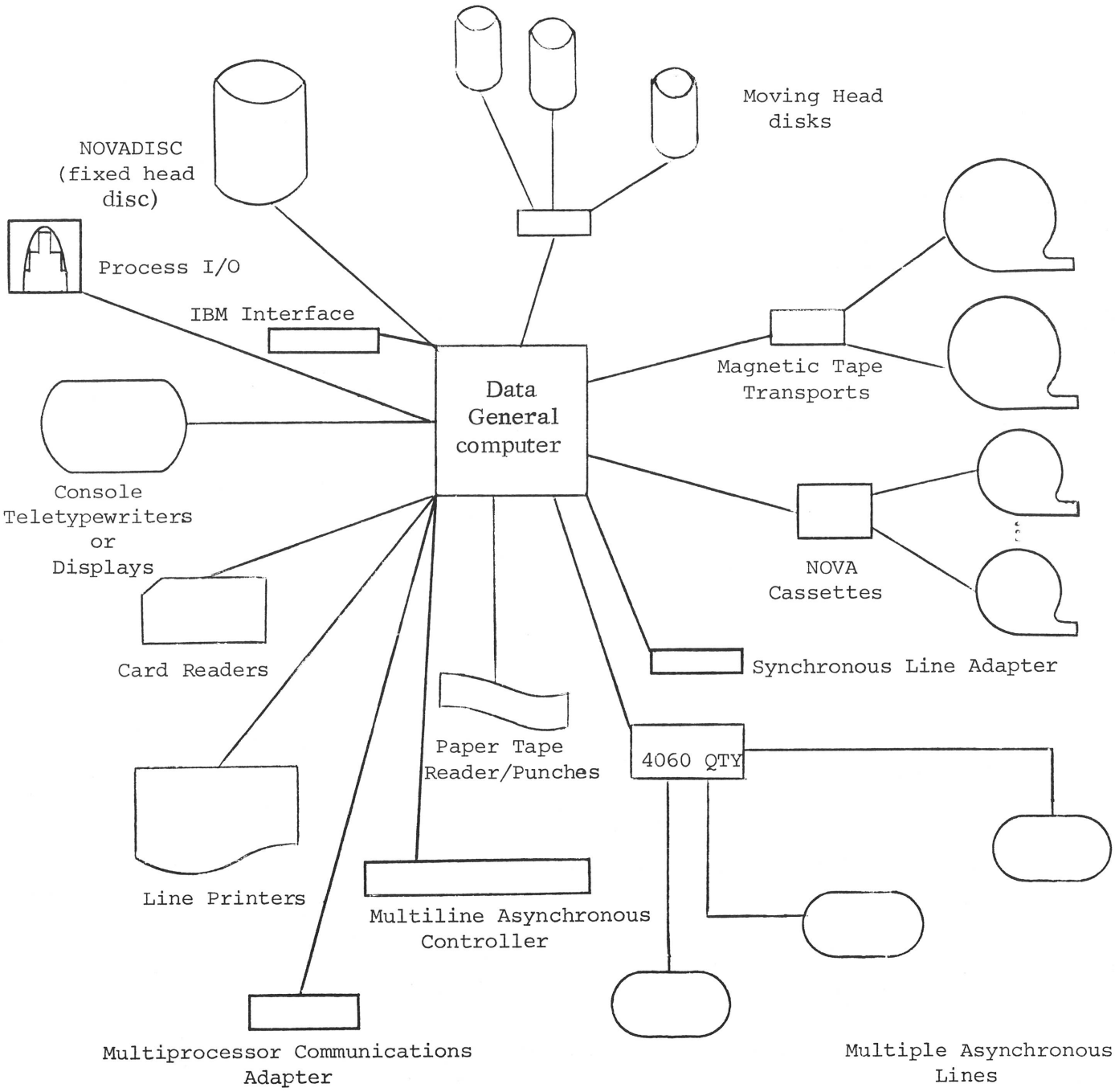
Word	Mnemonic	Purpose
0	DCTBS	Address of 8 word state save area (RTOS only)
1	DCTMS	Interrupt service mask
2	DCTIS	Device Interrupt Service Routine Address

USER INTERRUPT PROCESSING

Enough said about standard I/O devices. What about non-standard devices, the type customers are always most interested in? For these devices, a simple software interface exists to attach non-standard user devices to RDOS. This interface is provided through an abbreviated DCT (the first three entries of a standard DCT) which supplies the address of an 8-word state save area (if this program is to be reloaded and run under RTOS at some later date), the hardware interrupt mask to be set while servicing the user interrupt, and the address of the interrupt servicing routine. The system will store the program counter, accumulators, carry, current hardware mask, etc. on the interrupt stack before transferring control to the interrupt service routine. RTOS, not having such a stack, requires an explicit 8-word save area. The interrupt service routine is written by the user, and contains all program code necessary for processing the interrupt.

A system call, .IDEF, is used to insert pointers to user DCTs into the interrupt vector table, identifying the device to the system. Up to ten decimal user devices can be identified to the system at any moment. Input parameters necessary for the call to .IDEF are the device code of the user device and the DCT address of the user written driver. To remove these entries from the table, the system call .IRMV is issued with the device code passed as a parameter.

It is possible to activate user tasks from the interrupt servicing routine. This is done by transferring a non-zero message from the interrupt servicing routine to a user task via the .IXMT task monitor call. If the task expecting such a message has issued a .REC call, the task will be put into the ready state by the .IXMT call. If .REC has not been issued, the .IXMT call simply posts the message and the interrupt service routine finishes its processing.



RDOS Hardware Configurations

MULTIPLE SYSTEM DEVICES AND UNITS

The Real Time Disk Operating System is capable of supporting multiple disk and tape storage units, and multiple data processing devices. Support is given to these system devices when the operating system is generated.

The master system device for RDOS can be either a fixed or a moving head disk. RDOS supports up to two fixed head NOVADISC® * controllers, each with up to eight logical units of 128K, 256K, 512K or 756K storage words. Up to eight moving head units (disk pack or cartridge type) can also be included in any system, with from 2 to 20 disk surfaces per unit; maximum total storage is 98.4 million words.

From 1 to 16 magnetic tape transports and from 1 to 16 cassette drives can also be supported by RDOS. Magnetic tape units must be set at high density (800 bpi), and both 7 and 9 track tapes are allowed. Individual files can be referenced on both magnetic tape and cassette units by specifying both the transport drive and file number. Up to 100 files can be referenced on each unit. To refer to a file via the CLI or from a user program, the file must be specified by a file number as part of the file name. An example of a CLI command reference to the second file on magnetic tape transport 7 is XFER MT7:1 TEST. This command would load the second file on transport 7 and transfer it to disk under the file name TEST. Free format I/O is also available, yet it cannot be used in CLI commands.

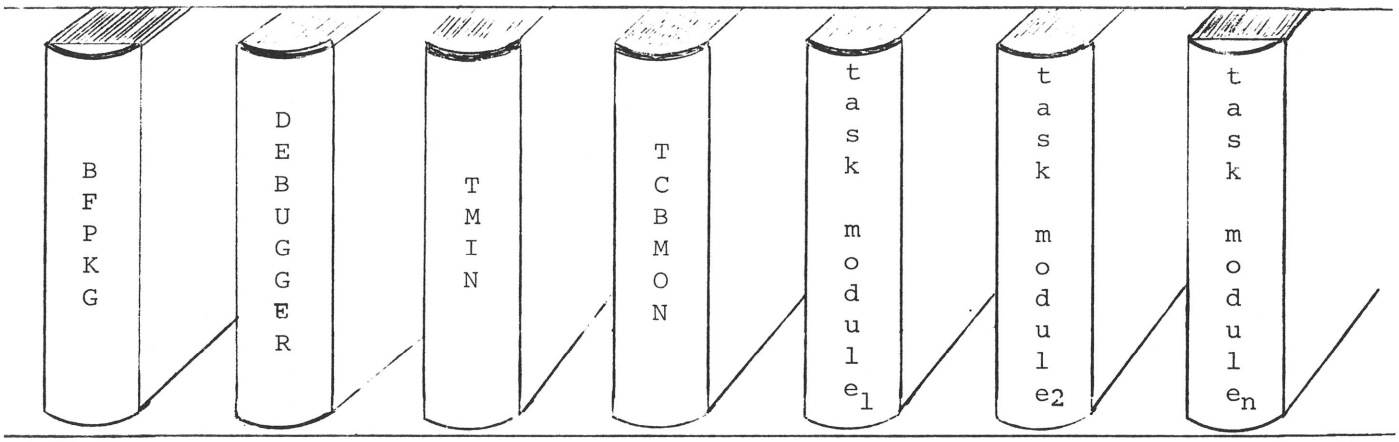
The type 4060 asynchronous data communications multiplexer (QTY) is another device which can be considered to be unit expandable. The QTY can accomodate from 1 to 64 full duplex lines at speeds up to 9600 baud. Each multiplexed line of the QTY corresponds to a file name of the form QTY:x, where x is the multiplexer line number in the range 0 to 64. Input/output operations are performed on each line by RDOS line/sequential reads and/or writes. RDOS permits unopened QTY lines to be monitored. If QTY:64 is opened and either a read line or a read sequential is attempted, the task issuing this call will be suspended until another, unopened line receives an interrupt. This line is then reported and the interrupting character is returned.

Since the system device drivers have been written reentrantly, multiple devices can be supported by RDOS with ease. This can be accomplished by simply adding another device control table (DCT) and by using existing device driver routines. Devices in this category are the console, paper tape reader/punch, line printer, card reader and incremental plotter.

Other non-system devices, e.g., A/D, Digital I/O interface, and Synchronous Line Adapter, can be given RDOS support as user devices. Documented drivers for these devices are given in other DGC publications.

---

*NOVADISC is a registered trademark of Data General Corporation, Southboro, Massachusetts.



RDOS System Library

BFPKG - I/O-to-core buffer package
DEBUGGER - Interrupt and non-interrupt debuggers
TMIN - Single task monitor
TCBMON - multitask monitor, .TASK and .KILL logic
Task Module$_1$ - .AKILL, .ASUSP, .ARDY logic
Task Module$_2$ - Task I.D. call logic
Task Module$_n$ - Other task processing logic

SYSTEM LIBRARY

The system library (SYS. LB) is a collection of program modules which support user programs run under RDOS. These modules can be likened to volumes on a library shelf. Each user program needing one or more of these modules selects them from the library by means of the Relocatable Loader, leaving behind those modules which are of no current use. By placing system modules in the library, a savings in user program core storage requirements thus results.

Among the volumes found in the RDOS system library are the multi and single task schedulers (TCBMON and TMIN), command processing modules for each task call type, the debuggers, and the Buffered I/O package (BFPKG) which has been discussed earlier.

Modules are extracted from this library by the relocatable loader. This loader is told which modules to load either by switches in the relocatable load CLI command or by means of the external normal pseudo-op, .EXTN. Except for the loading of a Task Scheduler, only those task modules will be loaded which are referenced by an .EXTN statement in the user program. The program must externally reference each task call name that is issued as a task call by the user program, or the loader will be unable to resolve the call. By only loading those task modules which are required for program operation a net savings in total core requirements is obtained. All modules taken from the library are loaded after the main (or root) program. Thus the loading of user programs and library modules occurs from low core to high core.

In addition to the debugger, task schedulers and other task modules, the Buffered I/O package (BFPKG) is also found. This module further enhances system operation by providing the user with asynchronous Line and Sequential data transfers, buffered in user program space.

- Command Line Interpreter
- Compile-Load-and-Go
- BATCH monitor
- Extended BASIC
- Extended Real-Time FORTRAN IV
- FORTRAN 5
- Extended Relocatable Assembler
- Macro Assembler
- Symbolic Text Editors
- Library File Editor
- Octal Editor
- Relocatable Loader
- Overlay Replacement Loader
- Symbolic Debugger

Real Time Disk Operating System Utilities

RDOS SUPPORTED UTILITIES

The Real Time Disk Operating System (RDOS) can be used for both the development and the implementation of user programs, and it allows these functions to be performed simultaneously under foreground/background programs. RDOS includes all the file capabilities normally available only on large machine disk operating systems, allowing the user to edit, assemble, execute, debug, compile, create and delete files.

Utilities currently supported under RDOS include the following:

1. Command Line Interpreter which acts as a keyboard interface to the operating system and its utilities.

2. Extended Assembler producing relocatable or absolute binary output from symbolic source programs. A macro assembler is also provided which has all the important features of the extended assembler and more, including a macro generation facility.

3. Relocatable Loader for linking relocatable binary files into an output core image (save) file. Also provided is an overlay replacement loader which can load and replace overlays selectively in an overlay file.

4. Extended FORTRAN IV with real time and file control extensions. A Compile-Load-and-Go capability also exists to facilitate the running of FORTRAN IV programs.

5. FORTRAN 5, an advanced high-level compiler, producing optimized object code and many language extensions.

6. Extended ALGOL compiler which provides many features in addition to those of ALGOL 60.

7. Several forms of single and multi-user Extended BASIC which permit the BASIC interpreter to run under RDOS.

8. A BATCH monitor which permits collections of user data processing jobs to be performed with minimum operator intervention.

9. Single-user Text Editors and multi-user text Editors for editing and updating program source files.

10. A Library File Editor enabling the user to separate, edit, and to maintain relocatable binary program libraries with ease. An Octal Editor enabling the user to examine and modify the contents of disk files.

11. Debug III which allows user programs to be debugged symbolically.

Glossary of Terms

Alias	Any name by which a file can be referred to which is other than the name given to the file when it was first created.
BATCH	A sequential processing procedure where items to be processed are collected into groups prior to processing. The groups are called "job streams" and all jobs in the job stream are processed within the same machine run.
Bootstrap	A technique for loading the first few instructions of a routine into storage, then using these instructions to bring in the remainder of the routine.
CLI (Command Line Interpreter)	A system program that accepts command lines from a system console and translates this input into commands for the utility programs.
COM.CM (See FCOM.CM)	A command file produced by the CLI to communicate to another utility or user program in the background.
Device	A hardware component of the system with unique operational characteristics.
Device Independence	The ability of a task to communicate with a device without regard to the unique nature of the device.
Device Space	The total addressable area of a mass storage device (disk, drum, etc.).
Directory	A file consisting of directory entries and directory access information. All user utilization of any file space is accomplished via a directory.
Directory Entry	A directory data element comprising 22 octal contiguous words. Each entry consists of an unpacked file name and file control/file access information.
Dormant State	Either the task has not been initiated (that is, made known to RDOS) or its execution was terminated or completed.
Executing State	The highest priority ready task has control of the central processing unit (CPU).

File	A collection of related data treated as a unit which is addressable by an alphanumeric identifier.
File Name	An alphanumeric file identifier. All file names within each directory must be unique. However, within a directory, one or more file names (a name and one or more aliases) can refer to the same file.
File Space	A subset of a device space. This subset is accessible to the file system modules.
FCOM.CM (see COM.CM)	A command file produced by the foreground CLI to communicate to another utility or user program in the foreground.
Foreground/Background	An environment where two active programs are competing for CPU resources. The foreground program is generally of a more critical nature (e.g., it performs real time control) than the background program, although this is not necessarily so.
K	An abbreviation for the number "1024" or "2^{10}". Thus "32K" words of memory are precisely 32,768 words.
Link Entry	A type of directory entry which points to another entry in some directory. This is analogous to an indirect address reference.
Multitasking	The ability to support more than one active task within an address space.
NMAX	The highest normally relocatable location used in a core image.
NREL	A term used to indicate that a program segment is normally relocatable. This means that after being loaded by the relocatable loader, the program segment will reside somewhere within the bounds of an area starting at address 400 octal and extending up to NMAX.
Overlay	Overlaying is a technique used for bringing routines (called "overlays") into main memory from some type of mass storage during the execution of a program. Overlaying permits several overlay routines to occupy the same main memory storage locations at different times. This technique is used when main memory is not large enough to satisfy a program's total storage requirements.

Overlay Node	The starting address of a main memory area that will be used to contain an overlay when it is brought in from a mass storage device.
Partition	A contiguous portion of file space which is less than or equal to total file space.
Primary Partition	A partition containing the total file space of a device.
Program	The contents of a complete address space.
Ready State	A condition in which a task is available for execution while a higher priority task has actual control of the CPU. Several tasks may be in the ready state at any one moment, but only the highest priority ready task will receive control of the CPU.
Resolution Entry	A directory entry which terminates a link entry chain.
Save File	A binary core image disk file of an executable program.
Secondary Partition	A partition containing some subset of a primary partition.
Spooling	A technique of temporarily saving data on a high speed device for output by the operating system to a slower peripheral.
Subdirectory	A directory which has an entry in a superior directory.
Subpartition	Same as "Secondary Partition."
Suspended State	A condition in which a task is awaiting the occurrence or completion of a system call, a task call, or some real time event.
Sysgen	A procedure used to customize an RDOS system to the available hardware and the expected user application environment.
System Call	A request to the operating system for use of system resources. All system calls are preceded by the mnemonic .SYSTM.
Task	A unique execution path within an address space.

Task Call	A request to the task monitor to effect a task state change. This also causes the task scheduler to pass control to the highest priority ready task.
Task Monitor	A collection of subroutine modules used to schedule and manage calls from user tasks.
Task State	The status of a task in the RDOS environment. A task may exist in one of four states: dormant, ready, suspended, and executing.
TCB (Task Control Block)	A block of memory which contains a task's state variable information and control information.
USP (User Stack Pointer)	A page zero location that is always preserved when control is switched from one task to another. This location is useful when multiple tasks share reentrant routines.
UST (User Status Table)	A 30 word area starting at address 400 octal which records all information pertinent to the execution of the entire user program.
ZMAX	The highest page zero location used in a core image.
ZREL	A term used to indicate that a program segment can be loaded anywhere within page zero user address space. That is, after the relocatable load process, the program segment may reside anywhere from address 50 octal to ZMAX.

Real Time Disk Operating System Support Literature

17-000001, Synchronous Communications Package. This is an application note which describes in depth a general purpose software package which can be used to control the SLA.

17-000002, User Device Driver Implementation in RDOS. This is an applications note which describes in depth the techniques which are required to add a device driver to RDOS, whether on a user level or on a system level.

17-000003, Buffered I/O Package in RDOS/RTOS. This is an applications note describing BFPKG, a module in SYS.LB, which provides asynchronous data buffering in main memory for user programs. BFPKG can be used by both RTOS and RDOS.

17-00004, Remote Synchronous Terminal Control Program. This application note describes how the Remote Synchronous Terminal Control Program, RSTCP, allows a Data General computer with peripherals to be operated as a remote intelligent data terminal. RSTCP is supported by both RDOS and RTOS.

017-000005, Multiline Asynchronous Controller Software Package. This applications note describes a general purpose subroutine package which is used to control the operation of a multiline asynchronous multiplexer. This multiplexer may be used with one or more Data General computers to control up to 1024 lines. The subroutine package may be run under RTOS or RDOS.

17-000008, IBM 360/370 Interface Software Package. This document describes a general purpose software subroutine package which can be used with either RTOS or RDOS to control the operation of the Type 4025 IBM 360/370 interface and a Data General Computer. The Type 4025 interface combines a general purpose hardware control unit and a software driver package. General descriptions of IBM 360/370 channel control procedures and timing are presumed to be known by the user before consulting this reference.

17-000013, Synchronous Communications Controller Software Package. This document describes a general purpose subroutine package which can be used to control the type 4015 Synchronous Communications Controller (SCC). This package is implemented as a procedure-independent runtime device driver to run under either RTOS or RDOS. The 4015 device enables any Data General computer to communicate with either a remote station or another computer. It provides a bi-directional full or half duplex interface at speeds up to 50K baud.

Real Time Disk Operating System Support Literature (Continued)

17-000014, Communications Multiplexer Software Package. This application manual describes a general purpose subroutine package which can be used to control the Type 4060/4073 Communications Multiplexer (CMS). The CMS package provides reentrant multitasking software that can be tailored to the characteristics of each line. Moreover, line and terminal procedures are implemented with ease, and optional modem control capabilities are available. The CMS package can be run under either RTOS or RDOS.

93-000018, Symbolic Text Editor Manual. This manual describes the use and operation of the symbolic text string editor under RDOS. This editor is needed to produce and correct source files for assembly, compilation, etc.

93-000040, Extended Assembler Manual. This user manual describes the use and operation of the extended relocatable assembler under RDOS.

93-000044, Debug III User's Manual. This document explains the use of the symbolic debuggers under RDOS; some of these debuggers disable interrupts, others do not.

93-000052, ALGOL User's Manual.

93-000053, FORTRAN IV User's Manual. This document describes DGC FORTRAN IV, including an exposition of its real time extensions.

93-000056, Real Time Operating System Reference Manual. This manual is the primary document to be consulted by users of RTOS, the core-only compatible subset of RDOS.

93-000065, BASIC User's Manual.

93-000068, FORTRAN IV Run Time Library User's Manual. This document describes the FORTRAN IV run time library routines in detail, and describes methods for interfacing these routines to assembler language programs.

93-000074, Library File Editor Manual.

93-000075, Real Time Disk Operating System Reference Manual. This is the primary document to be consulted by users of the Real Time Disk Operating System.

93-000080, Extended Relocatable Loaders Manual.

Real Time Disk Operating System Support Literature (Continued)

93-000081, Macro Assembler Manual. This manual describes the RDOS Macro Assembler. This assembler's functions are a compatible superset of those provided by the Extended Relocatable Assembler.

93-000083, Introduction to the Real Time Disk Operating System.

93-000084, Octal Editor Manual. This manual describes an RDOS utility used to examine or modify RDOS disk file space.

93-000085, FORTRAN 5 User's Manual.

93-000087, BATCH User's Manual. This document describes the features and operating procedures for the BATCH monitor.

93-000092, Stand-alone Disk Editor Manual. This manual describes the use of a disk editor which can be used to examine or modify all of disk space. A large portion of this manual is devoted to a discussion of directory structures of RDOS.

93-000096, FORTRAN 5 Run Time Library User's Manual. This manual describes the FORTRAN 5 run time library routines and describes methods for interfacing these routines to assembler language programs.

93-000105, RDOS User's Handbook. This manual summarizes the commands used with the Command Line Interpreter, as well as those commands used with each of the major RDOS utilities.

93-000106, Summary of Available Software. This manual describes the components of each license and model, and provides order forms for users wishing to order additional software.

93-000107, FORTRAN Commercial Subroutines Package. This manual describes a package which can be used by FORTRAN programmers who wish to write commercial application programs in FORTRAN.

93-000109, RDOS Console Reference Manual. This manual is the primary document to be consulted by users of the RDOS Command Line Interpreter.

93-000110, Software Summary and Bibliography. This manual summarizes the basic types of software provided by Data General Corporation, and provides a comprehensive bibliography of software documentation.

Real Time Disk Operating System Support Literature (Continued)

93-000111, SUPEREDIT Reference Manual. This manual describes an improved single-user symbolic text editor provided for use with Data General software.

*****

DataGeneral

SOFTWARE DOCUMENTATION
REMARKS FORM

Document Title	Document No.	Tape No.

SPECIFIC COMMENTS: List specific comments. Reference page numbers when applicable. Label each comment as an addition, deletion, change or error if applicable.

GENERAL COMMENTS: Also, suggestions for improvement of the Publication.

FROM:

Name	Title	Date

Company Name

Address (No. & Street)	City	State	Zip Code

FOLD DOWN FIRST FOLD DOWN

FIRST
CLASS
PERMIT
No. 26
Southboro
Mass. 01772

BUSINESS REPLY MAIL

No Postage Necessary If Mailed In The United States

Postage will be paid by:

Data General Corporation

Southboro, Massachusetts 01772

ATTENTION: Software Documentation

FOLD UP SECOND FOLD UP

STAPLE